SYBEX

Programmierung des 68000

Das umfassende Handbuch zum 68000-Prozessor

Der 68000 ist einer der ersten aus der Generation der 32-bit-Mikroprozessoren. Er bildet das Herz von leistungsstarken Computern wie Atari ST, Macintosh und Sinclair QL. Mit diesem Buch lernen Sie ohne Schwierigkeiten seine Möglichkeiten kennen und nutzen. Systematisch und schrittweise werden Sie mit dem 68000 vertraut gemacht:

Im ersten Teil wird die Struktur des Mikroprozessors beschrieben. Der zweite Teil stellt Ihnen Aufbau des Speichers, Adreßmodi und den Befehlssatz vor. Das Schlußkapitel gibt einen Überblick über die anderen Mitglieder der Prozessorfamilie: 68008, 68010, 68012 und 68020.

Zahlreiche Beispiele helfen Ihnen, das erworbene Wissen zu überprüfen und direkt praktisch anzuwenden.

Über die Autorin:

C. Vieillefond hat ihr Diplom an der Fachhochschule für Ingenieurwesen, Elektronik und Elektrotechnik gemacht. Sie hat bereits zahlreiche Fachartikel veröffentlicht.

3-88745-060-4

Vieillefond

Programmierung des 68000



3060

SYBEX

Programmierung des 68000



C. Vieillefond

Anmerkungen:
68000 ist ein geschütztes Warenzeichen von Motorola

Originalausgabe in Französisch
Titel der Originalausgabe: MISE EN OEUVRE DU 68000

Deutsche Übersetzung: Iris Cantarellas

Umschlaggestaltung: Jean-François Pénichoux/tgr
Satz: tgr – typo-grafik-repro gmbh, Remscheid
Gesamtherstellung: Boss-Druck und Verlag, Kleve

ISBN 3-88745-060-4
1. Auflage 1985

Printed in Germany

Inhaltsverzeichnis

Kapitel 1
Grundlagen

Der Prozessorchip 68000 präsentiert sich uns in einem Gehäuse mit 64 Anschlußstiften. Die erhöhte Anzahl der Anschlußstifte (im Vergleich zu anderen Mikroprozessoren) kommt einerseits dadurch zustande, daß die Daten- und Adreßbusse getrennt ausgeführt sind (es wird daher kein Multiplexing benötigt), und andererseits dadurch, daß dieser Mikroprozessor mit einer sehr vollständigen Signalpalette ausgestattet ist.

Das Anschlußschema dieses Prozessors ist in Abb. 1.1 zu sehen.

DIE SIGNALE DES MC68000

Die Ein- und Ausgänge des Gehäuses können nach Funktionen gruppiert werden, wie es das Diagramm der Abb. 1.2 zeigt.

Drei Kategorien von Anschlüssen erscheinen in diesem Diagramm:

1. die Stromversorgungen und der Takteingang;
2. der Adreß- und der Datenbus;
3. die Steuersignale, die außerdem noch 6 Gruppen bilden.

Bevor eine Untersuchung der Funktionen der Anschlüsse im einzelnen

Die Stromversorgungen: (2) VCC und (2) GND

Der MC68000 benötigt eine einzige Stromversorgung von 5 Volt. Es fällt jedoch auf, daß die Anschlußstifte VCC und GND doppelt ausgeführt und so zentral plaziert sind, daß aufgrund der daraus resultierenden kurzen Leitungswege innerhalb des Gehäuses die Stromverluste minimal gehalten werden.

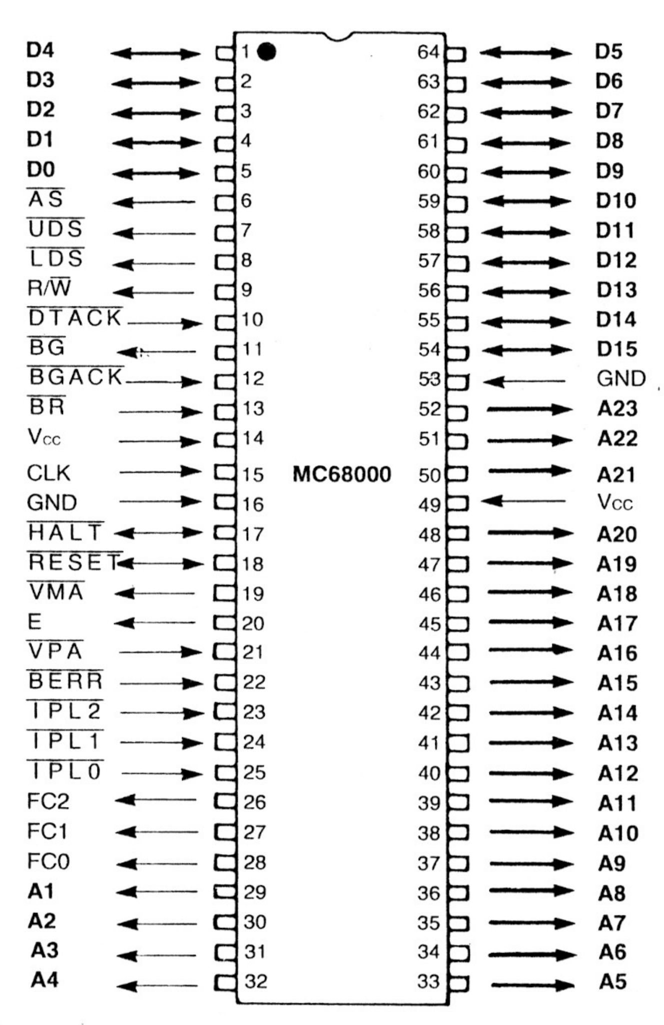


Abb. 1.1: Die Pinbelegung des MC68000

Der Takteingang: CLK

Das Eingangssignal des Taktgebers muß TTL-kompatibel und symmetrisch sein; es wird vom Mikroprozessor für das Generieren der internen Uhr benötigt.

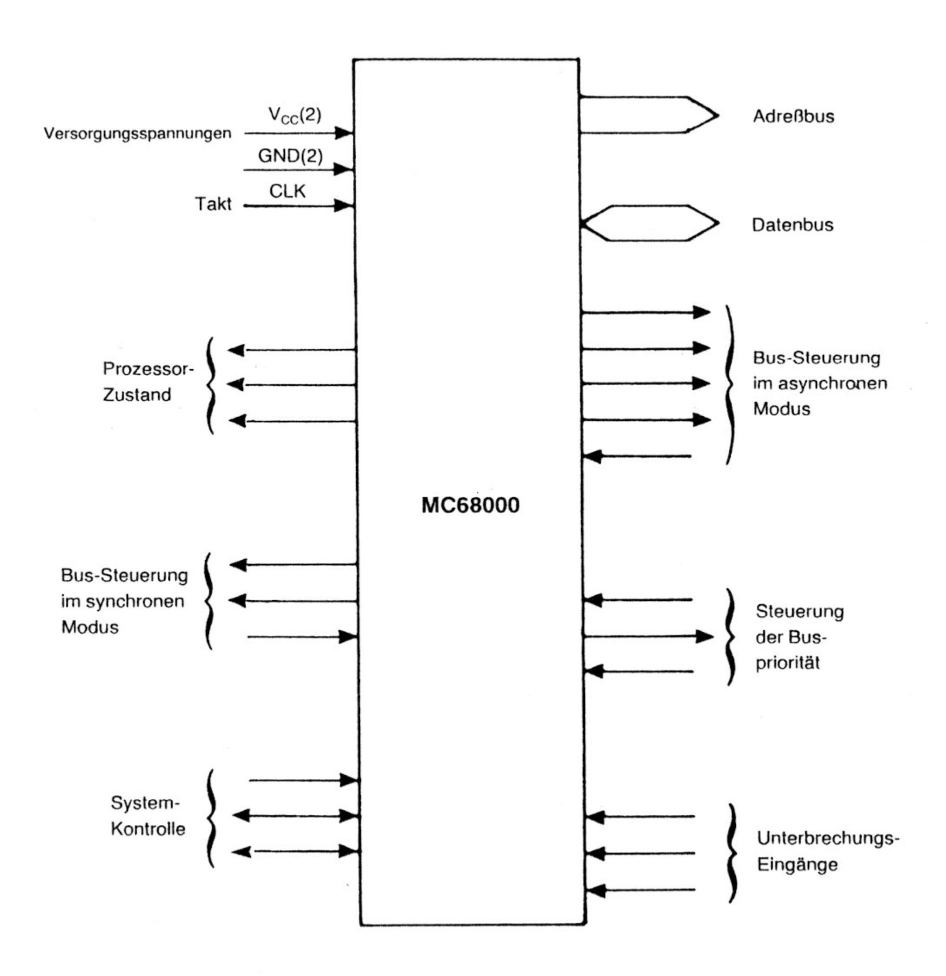


Abb. 1.2: Die Funktionen des MC68000

1. Die niedrigste Frequenz ist 2 MHz.

2. Die Anstiegs- und Abfallzeiten dürfen höchstens eine maximale Zeitdauer von 10 ns haben.

3. Für die unterschiedlichen Versionen des MC68000 (L4: 4 MHz, L6: 6 MHz, L8: 8 MHz, L10: 10 MHz, L12: 12.5 MHz) definieren wir eine minimale und maximale Impulslänge.

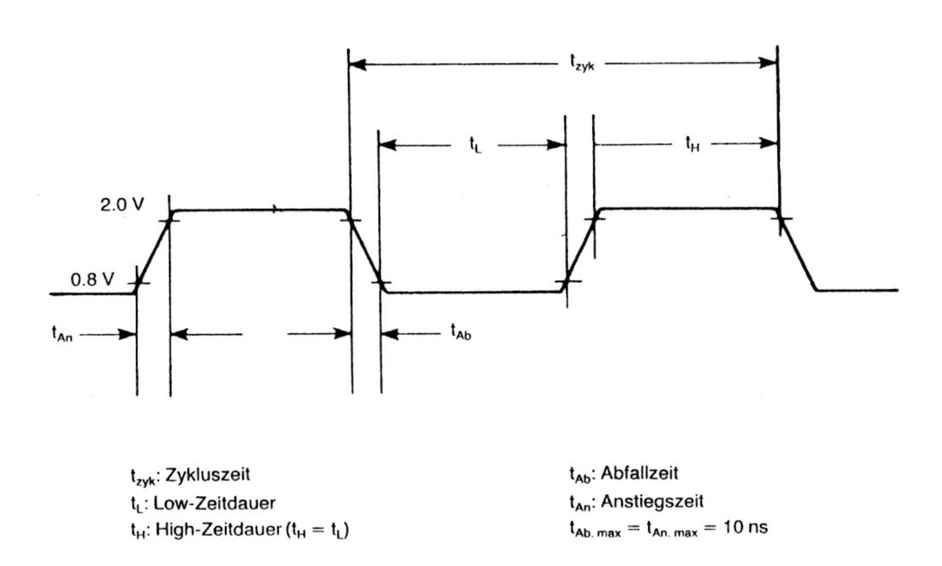


Abb. 1.3: Die Form des Taktgebersignals

Die Form des Taktgebersignals ist in Abb. 1.3 zu sehen.

Die minimale Zykluszeit der Version L8 ist 128 ns.

Im allgemeinen kann ein 8-MHz-Taktgeber mit einer angelegten Taktfrequenz von 16 MHz generiert werden. Wir schauen uns einmal eine solche Schaltung zur Generierung eines 8-MHz-Taktgebers an (Abb. 1.4) und

fassen dann die Taktgeber-Merkmale für die verschiedenen Versionen des 68000 zusammen (Abb. 1.5 und 1.6).

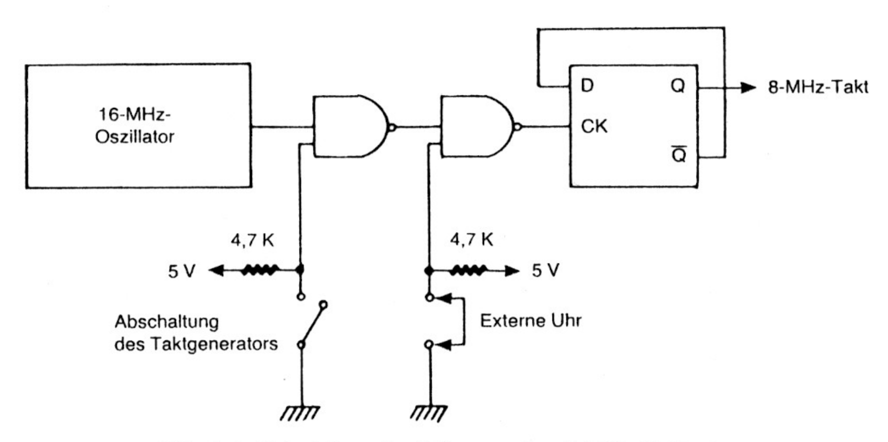


Abb. 1.4: Beispiel zur Realisierung eines 8-MHz-Taktgebers

	Abkürzungen	4 MHz MC68000 L4		6 MHz MC68000 L6		8 MHz MC68000 L8		10 MHz MC68000 L10		12,5 MHz MC68000 L12		Einheiten
Frequenz	F	2	4	2	6	2	8	2	10	4	12,5	MHz
Zykluszeit	t_{zyk}	500	250	500	167	500	125	500	100	250	80	ns

Abb. 1.5: Tabelle der Zykluszeiten abhängig von den Frequenzen der verschiedenen Versionen

	Abkürzungen	4 MHz MC68000 L4		6 MHz MC68000 L6		8 MHz MC68000 L8		10 MHz MC68000 L10		12,5 MHz MC68000 L12		Einheiten
		MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX	
Zykluszeit	t_{zyk}	250	500	167	500	125	500	100	500	80	250	ns
Impulslänge	t_L	115	250	75	250	55	250	45	250	35	125	ns
	t_H	115	250	75	250	55	250	45	250	35	125	
Anstiegs- und	t_{An}	–	10	–	10	–	10	–	10	–	5	ns
Abfallzeit	t_{Ab}	–	10	–	10	–	10	–	10	–	5	

Abb. 1.6: Tabelle der minimalen und maximalen Zeiten

Der Datenbus: D0-D15

Der Datenbus ist ein 16-Bit-Datenbus, d. h. er kann 8 oder 16 Datenbits parallel (also gleichzeitig) übertragen. Er kann drei Zustände annehmen (tristate) und ist außerdem bidirektional, d. h. er überträgt in beide Richtungen.

Der Adreßbus: A1-A23

Der Adreßbus ist ein 23 Bit breiter, unidirektionaler Tristate-Bus, der 8 Mega Datenworte (2^{24} entsprechen 16 Megabyte) adressieren kann.

Es fällt auf, daß dieser Bus keinen Ausgang mit der Bezeichnung A0 hat. A0 wird durch 2 Signale $\overline{UDS}$ (Upper Data Strobe) und $\overline{LDS}$ (Lower Data Strobe) ersetzt, die abhängig vom Typ der adressierten Daten vom 68000 angesprochen werden. Diese Signale werden im folgenden Abschnitt näher besprochen.

Steuersignale des Busses im Asynchron-Modus: $\overline{AS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{LDS}$, $\overline{DTACK}$

Im asynchronen Modus arbeitet der Prozessor wie folgt: Nachdem der Mikroprozessor eine Anfrage für eine Datenübertragung an ein peripheres Gerät geschickt hat, wartet er auf eine Antwort von dem angesprochenen Gerät. Der Prozessor stellt sich also auf die Geschwindigkeit des Peripheriegerätes oder des Speichers, mit dem er kommunizieren soll, ein.

Die Signale, die für diesen Typ des Datenaustauschs benötigt werden, sind die folgenden:

$\overline{AS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{LDS}$, $\overline{DTACK}$

$\overline{AS}$ *Address Strobe/Adreßimpuls*

Dieses Signal weist darauf hin, daß der Bus augenblicklich eine gültige Adresse enthält.

R/$\overline{W}$ *Read-Write/Lesen-Schreiben*

Dieses Signal definiert den Übertragungstyp der Daten auf dem Bus, also entweder Lesen oder Schreiben.

$\overline{UDS}$, $\overline{LDS}$ *Upper, Lower Data Strobe/Impuls höher- bzw. niederwertiger Daten*

Diese Signale werden intern im MC68000 generiert und sind abhängig vom Datentyp, der übertragen werden soll, und von A0. In Verbindung mit dem $R/\overline{W}$-Signal bestimmen $\overline{UDS}$ und $\overline{LDS}$ den Lese- oder Schreibvorgang eines Wortes oder eines Bytes.

Bevor wir die unterschiedlichen Kombinationen dieser drei Signale betrachten (Abb. 1.7) wollen wir zunächst die Speicherorganisation näher betrachten und die Begriffe Wort (16 Bits) und Byte (8 Bits) klären.

	15 — 8	7 — 0
Wort 000000	Byte 0000000	Byte 000001
Wort 000002	Byte 0000002	Byte 000003
Wort 000004	Byte 0000004	Byte 000005
Wort FFFFFE	Byte FFFFFE	Byte FFFFFF
	Bytes an geraden Adressen	Bytes an ungeraden Adressen

$\overline{UDS}$	$\overline{LDS}$	$R/\overline{W}$	D8-D15	D0-D7	Operation
H	H	–	nicht gültig	nicht gültig	–
L	L	H	gültig: Bits 8-15	gültig: Bits 0-7	Lesen eines Wortes
H	L	H	nicht gültig	gültig: Bits 0-7	Lesen eines Bytes an ungerader Adresse
L	H	H	gültig: Bits 8-15	nicht gültig	Lesen eines Bytes an gerader Adresse
L	L	L	gültig: Bits 8-15	gültig: Bits 0-7	Schreiben eines Wortes
H	L	L	Kopie der Bits 0-7	gültig: Bits 0-7	Schreiben eines Bytes an ungerader Adresse
L	H	L	gültig: Bits 8-15	Kopie der Bits 8-15	Schreiben eines Bytes an gerader Adresse

Abb. 1.7: Kombinationstabelle der Signale $R/\overline{W}$, $\overline{UDS}$, $\overline{LDS}$

Im Diagramm der Abb. 1.8 ist der Schreibvorgang eines Bytes schematisch dargestellt. Das Zurückkopieren wird bei dieser Darstellung vernachlässigt, da diese Operation bei den jetzigen Generationen des 68000 nicht mehr vorgesehen ist.

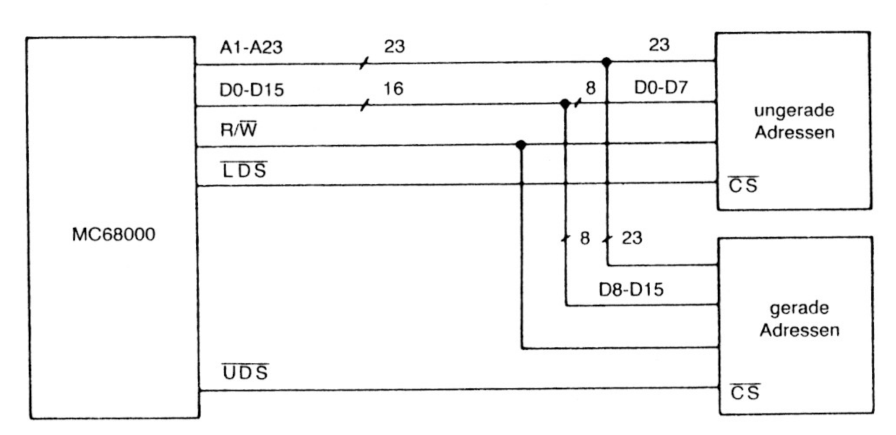


Abb. 1.8: Adressierung von Bytes

$\overline{\text{DTACK}}$ *Data Transfer ACKnowledge/Empfangsbestätigung der Datenübertragung*

Dieses Signal gibt dem Mikroprozessor die folgende Information:

Beim Lesevorgang: Die zu lesenden Daten sind auf dem Bus verfügbar.

Beim Schreibvorgang: Die Daten sind von einem externen Baustein assimiliert worden.

Steuersignale für die Zugehörigkeit der Busse: $\overline{\text{BR}}$, $\overline{\text{BG}}$, $\overline{\text{BGACK}}$

Diese Signale bestimmen, welche der externen Bausteine die Steuerung der Busse kontrollieren oder kontrollieren werden. Diese drei Steuersignale heißen:

$\overline{\text{BR}}$, $\overline{\text{BG}}$, $\overline{\text{BGACK}}$

$\overline{\text{BR}}$ *Bus Request/Busanfrage*

Dieses Signal weist den Mikroprozessor darauf hin, daß eine andere intelligente Einheit auf die Busse zugreifen möchte.

$\overline{BG}$ *Bus Grant/Busfreigabe*

Der anfragende Baustein wird durch dieses Signal darauf hingewiesen, daß der 68000 seine Anfrage $\overline{BR}$ berücksichtigt hat. Die Umgebung des Mikroprozessors wird davon am Ende des ablaufenden Buszyklus informiert (der Begriff Buszyklus wird später erklärt).

$\overline{BGACK}$ *Bus Grant ACKnowledge/Empfangsbestätigung des* $\overline{BG}$

Der anfragende Baustein bestätigt die Übernahme der Busse und ihre Steuerung. Dieses Signal kann nicht bestätigt werden ($\overline{BGACK}$ im Low-Zustand), wenn nicht die folgenden 4 Voraussetzungen erfüllt sind:

1. $\overline{BG}$ wurde empfangen und bestätigt vom Mikroprozessor.
2. $\overline{AS}$ ist inaktiv, was besagt, daß der Mikroprozessor die Busse nicht benutzt.
3. $\overline{DTACK}$ ist inaktiv, was besagt, daß kein Peripheriegerät oder externer Speicher mit dem zentralen Prozessor in Verbindung steht und die Busse benutzt.
4. $\overline{BGACK}$ ist inaktiv, also kein anderer intelligenter Baustein ist im Besitz der Buskontrolle.

Die Rolle dieser Signale ist also die rigorose Verwaltung des Multiprozessoren-Systems mit einer großen Störungsimmunität.

Steuersignale der Unterbrechungen: $\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$
Interrupt Pending Level

Diese drei Eingänge vermitteln dem Mikroprozessor, daß ein Peripheriegerät eine Unterbrechung fordert. Die Prioritäten sind durch eine codierte Zahl geregelt. Die Zahl 7 repräsentiert dabei die höchste Priorität, während 0 darauf hinweist, daß keine Unterbrechung erwartet wird.

Steuersignale des Systems: $\overline{RESET}$, $\overline{HALT}$, $\overline{BERR}$

Diese Leitungen werden zur Initialisierung und zur Unterbrechung des Prozessors benötigt. Außerdem weisen sie auf die Fehler, die während eines Datenaustauschs aufgetreten sind, hin. Diese Signale sind:

$\overline{BERR}$, $\overline{RESET}$, $\overline{HALT}$

$\overline{BERR}$ *Bus ERRor/Fehler auf dem Bus*

Dieser Eingang informiert den Prozessor über ein Problem, das sich während des Zyklusablaufs ergeben hat. Dieses Problem kann beispielsweise sein:

- ein Peripherigerät, das dem Prozessor keine Antwort gibt;
- bei der Vergabe der Interrupt-Vektornummer ist ein Fehler aufgetreten;
- eine ungültige Adresse ist erzeugt worden usw.

Dieses Signal ist entweder von einer externen Logik generiert oder aber sofort vom teilnehmenden Baustein geliefert worden. Zum Beispiel erzeugt der Baustein MC6841, eine Speicherverwaltungseinheit, ein FAULT-Signal, wenn ein nicht definiertes Segment angesprochen wird. In diesem Fall wird das FAULT-Signal sofort am $\overline{\text{BERR}}$-Eingang des MC68000 angelegt.

$\overline{\text{RESET}}$
Dieses Signal ist bidirektional.

Im Eingabemodus: Der Prozessor wird als Antwort auf ein externes Signal initialisiert.

Im Ausgabemodus: Die Ausführung des RESET-Befehls bewirkt die Zurücksetzung aller externen Bausteine in ihren Anfangszustand. In diesem Fall wird der Zustand des Prozessors nicht geändert.

Bemerkung: Die gesamte Initialisierung (Prozessor und externe Bausteine) wird durch die gleichzeitige Verwendung der Signale $\overline{\text{HALT}}$ und $\overline{\text{RESET}}$ auf den Eingangsanschlußstiften herbeigeführt.

$\overline{\text{HALT}}$
Dieses Signal ist ebenfalls bidirektional.

Im Eingabemodus: Wenn dieses Signal durch einen externen Baustein erzeugt wird, bringt es den Prozessor am Ende des aktuellen Buszyklus zum Stillstand. Alle Steuersignale werden inaktiv und alle Leitungen in den hochohmigen Zustand versetzt.

In Verbindung mit dem $\overline{\text{RESET}}$-Signal bewirkt das $\overline{\text{HALT}}$-Signal die Initialisierung des gesamten Systems.

Im Ausgabemodus: Der Prozessor unterbindet die Ausführung der Befehle, nachdem beispielsweise ein doppelter Fehler auf dem Bus festgelegt worden ist. Das $\overline{\text{HALT}}$-Signal weist die Peripheriegeräte nun auf den Funktionsstillstand hin. Die einzige Möglichkeit, den $\overline{\text{HALT}}$-Zustand zu verlassen, ist, einen Impuls auf der $\overline{\text{RESET}}$-Leitung an den Prozessor auszusenden.

Steuersignale der Peripheriegeräte der 6800-Serie: E, $\overline{VPA}$, $\overline{VMA}$

Diese Steuerleitungen ermöglichen die Schnittstellen-Verbindung zwischen den im Synchron-Modus arbeitenden Peripheriegeräten der 6800-Familie und dem MC68000; wir erinnern uns, daß der MC68000 im Asynchron-Modus arbeitet. Diese Signale heißen:

E, $\overline{VPA}$, $\overline{VMA}$

E *Enable/Freigabesignal (ist ein Ausgangssignal)*

Dies ist das Standardsignal für die Kommunikation mit den 6800-Bausteinen. Es beansprucht die zehnfache Zeit wie der Eingangstakt des MC68000 und ermöglicht die Zusammenarbeit des MC68000 mit den Peripheriegeräten im Synchron-Modus.

$\overline{VPA}$ *Valid Peripheral Address/Gültige Peripherie-Adresse*

Dieser Eingang bewirkt die Synchronisation des Datentransfers über E, wenn ein Peripheriegerät im Synchron-Modus mit dem Prozessor kommunizieren möchten. Gleichzeitig deutet dieses Signal darauf hin, daß sich der Prozessor in eine vektorielle Unterbrechung begibt.

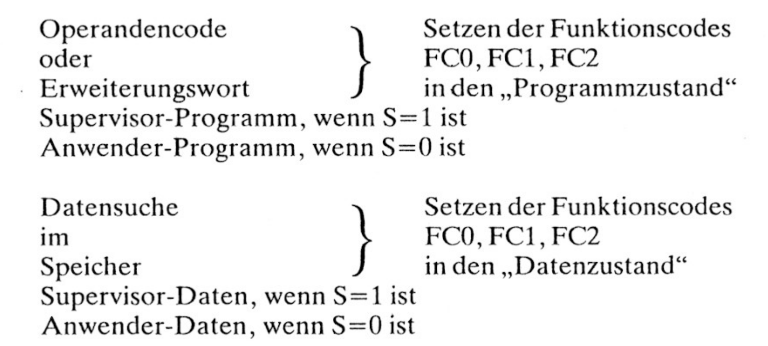
$\overline{VMA}$ *Valid Memory Address/Gültige Speicheradresse*

Ein Signal auf diesem Ausgang zeigt dem Peripheriebaustein der 6800-Familie, daß die Adresse auf dem Bus gültig ist und daß der MC68000 nun auf das E-Signal synchronisiert ist. Das Signal $\overline{VMA}$ stellt die Antwort auf das $\overline{VPA}$-Signal dar.

Betriebszustand des Prozessors: FC0, FC1, FC2
Function Code/Funktionscode

Diese drei Ausgänge deuten auf die Betriebszustände des Prozessors hin; er kann sich im Supervisor-Modus („nicht beschränktes" System) oder im Anwender-Modus („beschränktes" System) befinden. Außerdem weisen die drei Ausgänge auf den angewendeten Zyklustyp hin. Diese Information ist dann gültig, wenn das Signal $\overline{AS}$ anliegt.

Die Funktionscodes ändern sich mit jedem Buszyklus. Sie werden vom Prozessor nach Untersuchung des S-Bits (Bit des Statusregisters) und in Übereinstimmung mit dem auszuführenden Buszyklus gesetzt:



Beispiel:

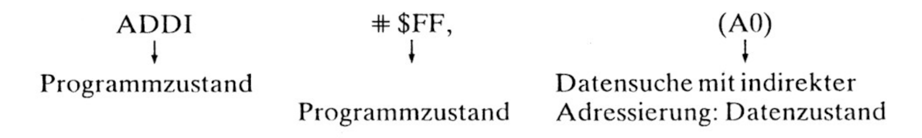


Tabelle der Funktionscodes:

FC2	FC1	FC0	Zyklustyp
L	L	L	reserviert
L	L	H	Anwender-Daten
L	H	L	Anwender-Programm
L	H	H	reserviert
H	L	L	reserviert
H	L	H	Supervisor-Daten
H	H	L	Supervisor-Programm
H	H	H	Interrupt ist aufgetreten

Anwendung dieser Funktionscodes:

Diese Codes kontrollieren beispielsweise die unterschiedlichen Zugriffsarten oder aber legen eine Differenzierung gewisser spezieller Zustände des Prozessors fest, wie etwa den Nachweis eines Interrupts (Unterbrechung).

Beispiel für die Zugriffskontrolle:

Der Supervisor-Speicher wird im Falle eines Anwenderzugriffs, wenn FC2=1 ist, dadurch geschützt, daß ein spezieller Supervisor-Bereich adressiert wird (Abb. 1.9).

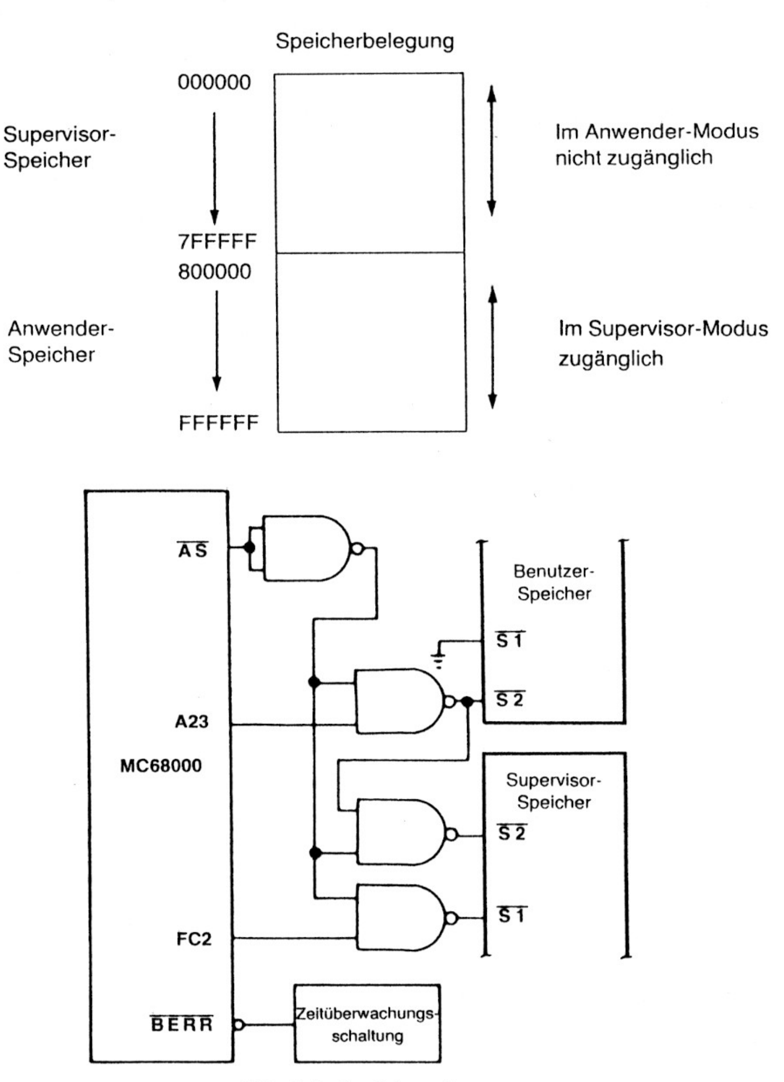


Abb. 1.9: Speicherschutz

Ein weiteres Beispiel für die Gültigkeit des Speicherbereichs:

In diesem Fall wird die Bestätigung der verschiedenen Speicherbereiche über die Funktionscodes FC0, FC1, FC2 geregelt (Abb. 1.10).

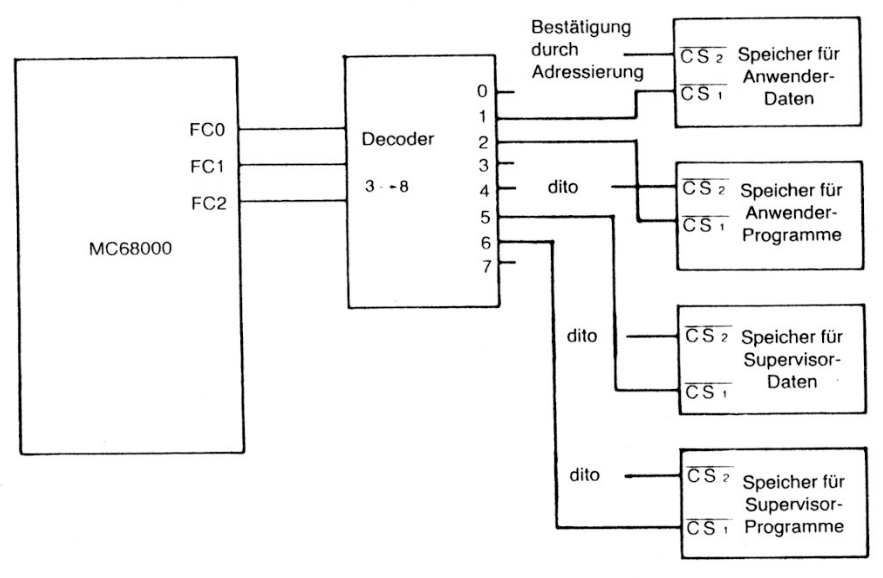


Abb. 1.10: Beispiel für die Bestätigung der Speicherplatzbereiche

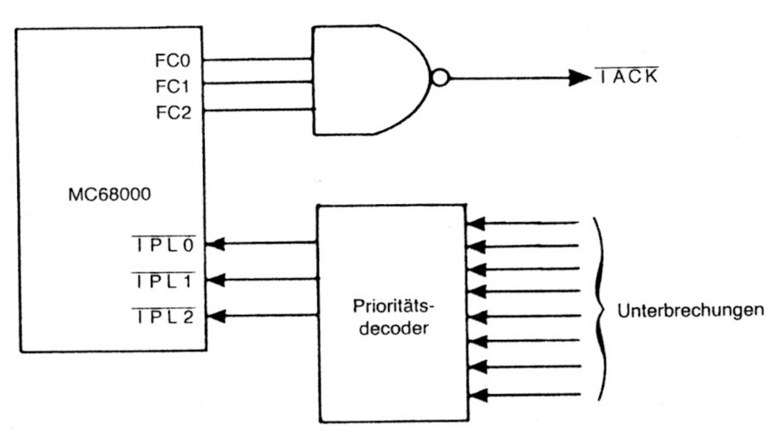


Abb. 1.11: Schaltung zur Unterbrechungsbestätigung mit Hilfe der Funktionscodes

Beispiel für den Zustand des Prozessors bei Auftreten eines Interrupts:

Wenn der Prozessor in die Phase der Interruptbestätigung eintritt, werden die Codes FC0, FC1, FC2 geändert und zur Erzeugung des Empfangsbestätigungssignals der Unterbrechung $\overline{IACK}$ verwendet (Abb. 1.11).

DER INTERNE AUFBAU DES MC68000

Zunächst wollen wir uns die Bedeutung der folgenden Fachbegriffe wieder ins Gedächtnis rufen:

- ein Byte – 8 Bits
- ein Wort – 16 Bits
- ein Langwort oder Doppelwort – 32 Bits

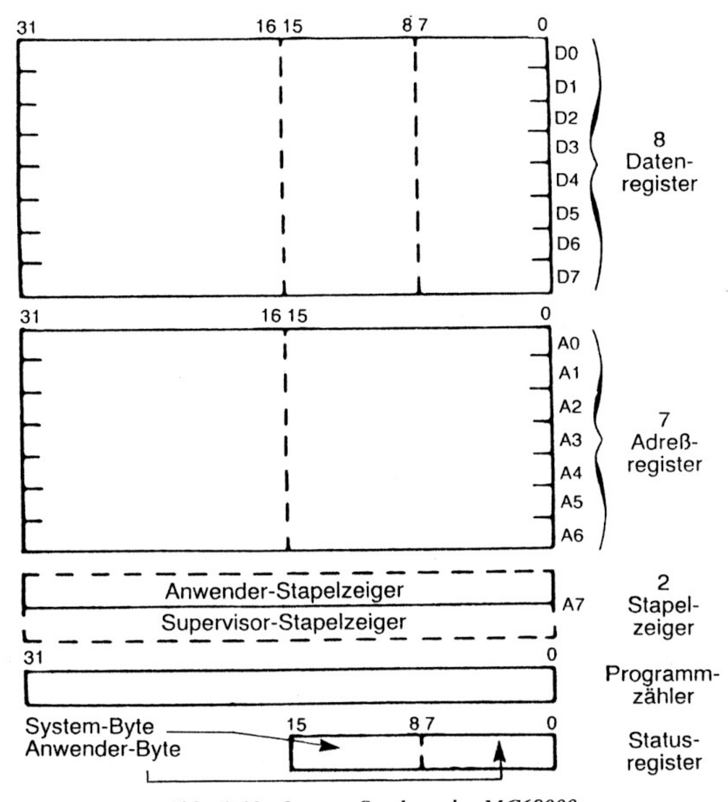


Abb. 1.12: Interne Struktur des MC68000

Es folgt nun die Liste der Register unseres Mikroprozessors:

– 8 32-Bit-Datenregister: D0, D1, ..., D7
– 8 32 Bit-Adreßregister: A0, A1, ..., A7
– ein 32-Bit-Programmzähler: PC
– ein 16-Bit-Statusregister: SR

Das Statusregister setzt sich aus zwei Bytes zusammen. Jedes Bit dieses Registers hat eine ganz bestimmte Bedeutung.

– Byte des niederwertigen Teils: Anwender-Byte
– Byte des höherwertigen Teils: System-Byte

Die interne Struktur des MC68000 ist durch das Schema der Abb. 1.12 dargestellt.

Beschreibung des Statusregisters

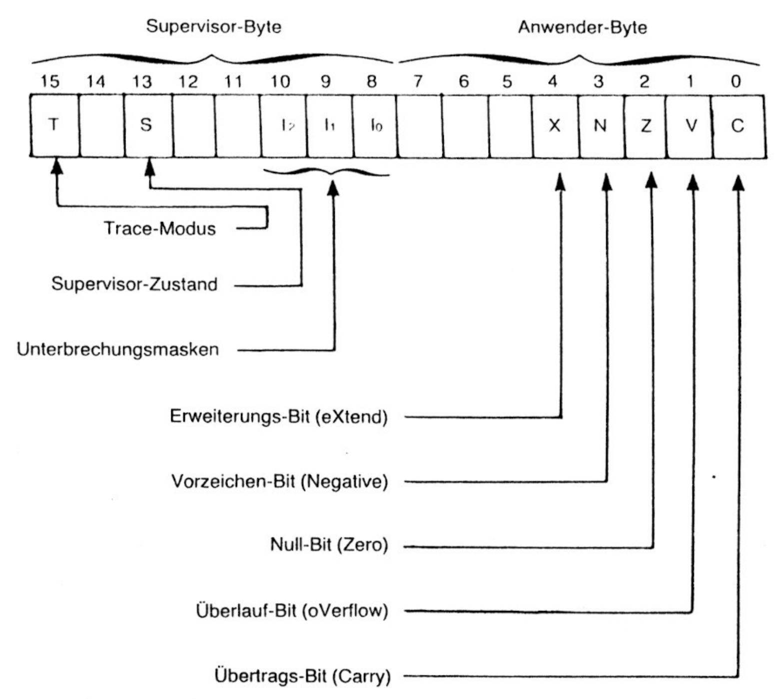


Abb. 1.13: Struktur des Statusregisters

Bemerkung: Auf Anwenderebene kann nur in das Anwender-Byte geschrieben werden. Dagegen kann im Supervisor-Modus in das ganze Wort geschrieben werden.

Bit T: Trace (Ablaufverfolgung)

Wenn dieses Bit gesetzt ist (T=1), befindet sich der Prozessor in der Betriebsart „Trace". Das bedeutet, daß das T-Bit nach jedem Befehl überprüft wird; wenn es den Wert 1 besitzt, wird zu einem vom Anwender geschriebenen Trace-Unterprogramm verzweigt. Dieses Trace-Unterprogramm kann zum Beispiel verwendet werden zur Prüfung des Inhalts von ausgewählten Registern oder Speicherplätzen, zur Statusprüfung oder zur Durchführung irgendwelcher anderer Fehlersuchaufgaben.

Ist das T-Bit nicht gesetzt, wird der nächstfolgende Befehl ausgeführt.

Das Prinzip der Trace-Betriebsart wird in Abb. 1.14 gezeigt.

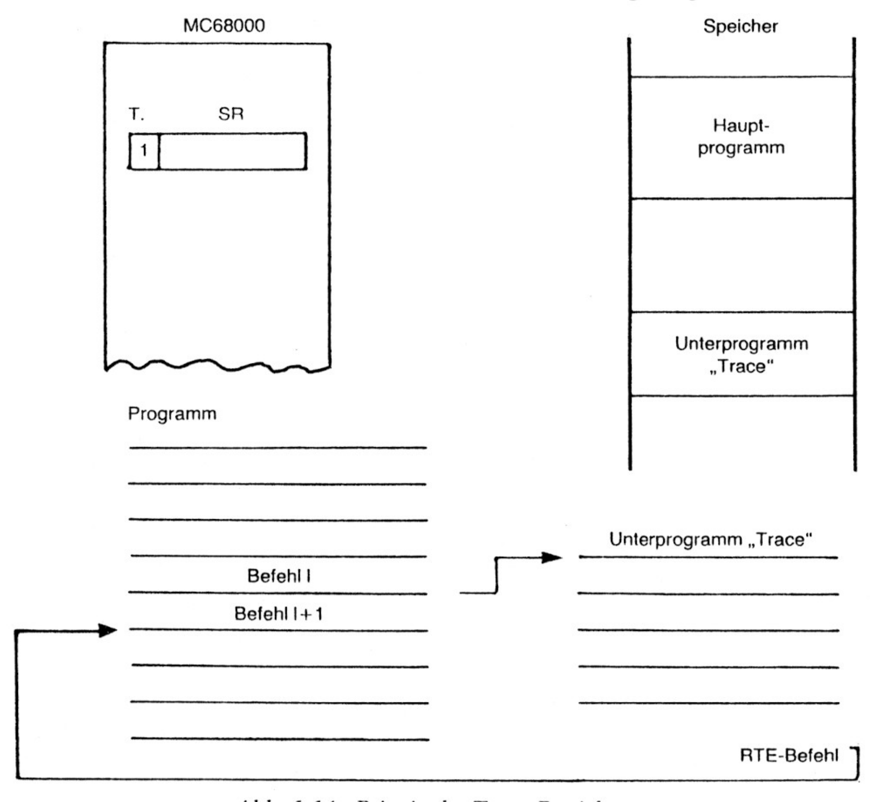


Abb. 1.14: Prinzip der Trace-Betriebsart

Bit S: Supervisor (Überwachungs- oder System-Bit)

Dieses Bit definiert den Modus, in dem der Prozessor im weiteren arbeiten wird:

- Benutzerebene S=0
- Supervisor- oder Systemebene S=1

Der Supervisor-Modus unterliegt keinerlei Einschränkungen, weder in der Befehlsskala noch in adressierbaren Bereichen.

Dagegen ist der Anwender-Modus einschränkend:

- Nicht alle Befehle sind erlaubt: Die sogenannten „privilegierten" Befehle sind verboten.
- Bestimmte Bereiche können nicht angesprochen werden. Beispielsweise ist das höherwertige Byte des Statusregisters SR nicht zugänglich; ebenso ist der Zugriff auf den Systemstapelzeiger SSP, Supervisor-Daten und -Programme verboten.

Bits I2, I1, I0: Interrupt mask (Unterbrechungsmaske)

Diese drei Bits enthalten die Unterbrechungsmaske. Sie bestimmen die Prioritätsebene der Unterbrechungsanforderungen. Die Unterbrechungen eines niedrigeren oder gleichwertigen Niveaus als das momentan gültige Unterbrechungsniveau bleiben jeweils unberücksichtigt. Nur die Unterbrechungen eines höheren Niveaus werden beachtet.

Es stehen 7 Prioritätsebenen zur Verfügung, wobei die Ebene 0 keine Priorität bedeutet, dagegen stellt das Niveau 7 die höchste Priorität dar.

Diese Masken werden automatisch während des Ablaufs der Unterbrechung gesetzt.

Mit Hilfe der folgenden Tabelle werden die Zustände dieser drei Bits sichtbar:

EBENE	I2	I1	I0	
7	1	1	1	→ höchste Ebene, nur die nicht maskierbaren Unterbrechungen können ausgeführt werden
6	1	1	0	
5	1	0	1	
4	1	0	0	
3	0	1	1	
2	0	1	0	
1	0	0	1	→ niedrigste Prioritätsebene
0	0	0	0	→ keinerlei Priorität

Bit N: Negative (Vorzeichen-Bit)

Dieses Bit wird auf 1 gesetzt, wenn eine arithmetische, logische, Schiebe- oder Rotieroperation zu einem negativen Ergebnis führt. Das N-Bit entspricht also dem höchstwertigen Bit des Ergebnisses.

Bit Z: Zero (Null)

Dieses Bit wird auf 1 gesetzt, wenn das Ergebnis einer Operation 0 ist.

Bit V: oVerflow (Überlauf)

Dieses Bit wird gesetzt, wenn ein arithmetischer Überlauf auftritt; dies bedeutet, daß das Ergebnis die Länge des Operanden überschritten hat.

Bit C: Carry (Übertrag)

Dieses Bit wird auf 1 gesetzt, wenn bei einer Addition ein Übertrag entsteht oder bei einer Subtraktion ein Entlehnwert benötigt wird.

Bit X: eXtend (Erweiterung)

Dieses Bit ist das Übertragsbit für Operationen mit erhöhter Genauigkeit. Im allgemeinen reflektiert es das C-Bit. Die Bedeutung des X-Bits wird am Anfang des Kapitels über Anwenderprogramme noch deutlich werden.

Das Register A7

Die Adreßregister können als Basisadreßregister, Softwarezeiger zu benutzerdefinierten Speicherbereichen, zur temporären Aufnahme von Adreßwerten und für den Zugriff auf Bytes, Worte oder Doppelworte im Speicher dienen. Dabei fällt dem Register A7 eine ganz spezielle Rolle zu. Dieses Register übernimmt die Funktion des Stapelzeigers. Es dient der Sicherung der Rückkehradresse im Fall eines Unterprogrammaufrufs und während des Ablaufs von Fehlerprozeduren. Abhängig vom S-Bit ist entweder das Register A7 oder A7' aktiv.

Ist nämlich das S-Bit des Statusregisters 0, ist der Teil A7 des Registers aktiv, der den Benutzer-Stapelzeiger, USP (User Stack Pointer) genannt, darstellt. Wenn das S-Bit den Wert 1 besitzt, ist das Register A7' angesprochen; es handelt sich hierbei um den Supervisor-Stapelzeiger SSP (Supervisor Stack Pointer).

Im folgenden geben wir nun einige Definitionen, die Sie mit dem Grundbegriff des Zyklus vertraut machen sollen.

DIE ZYKLEN

Clock cycle/Taktzyklus

Darunter versteht man die Zeit, die zwischen zwei aufeinanderfolgenden gleichartigen Zuständen verstreicht. Wenn das Eingangssignal 8 MHz besitzt, beträgt tzyk 125 ns.

Bus cycle/Buszyklus

Das ist die Zeit, die notwendig ist, um einen beliebigen Datenaustausch vorzunehmen:

- Lesen eines Bytes oder eines Wortes
- Schreiben eines Bytes oder eines Wortes
- Lesen/Ändern/Schreiben eines Bytes

Diese Zeit ist natürlich abhängig von den Zugriffszeiten der externen Bausteine.

Instruction cycle/Befehlszyklus

Hierbei handelt es sich um die Gesamtzeit, die benötigt wird, um die Durchführung eines Befehls zu verwirklichen: die Zeit der Buszyklen und die des Prozessors. Diese Zeitspanne ist abhängig vom Befehlstyp, den benötigten Buszyklen und dem „Prefetch"-Mechanismus („Vorgriff"-Mechanismus), d. h. manche Buszyklen werden vom Prozessor durch Vorziehen eines Befehls eingespart.

Der Beginn eines Befehlszyklus wird durch den Prozessor bestimmt und ist von seiner vorhergehenden Aktivität abhängig (Ende einer Datenübertragung, Ende einer internen Operation).

Im allgemeinen beanspruchen die Lese- und Schreibvorgänge 4 Taktzyklen. Aber wenn ein Wartezustand eintritt, kann ein Zyklus auch länger dauern.

Übungen

1.1: Wie viele Leitungen besitzt der Adreßbus?

1.2: Ist die Adresse eines Wortes gerade oder ungerade?

1.3: Hat das höherwertige Byte eines Wortes eine gerade oder eine ungerade Adresse?

1.4: Welche Operation wird ausgeführt, wenn folgende Signale gesetzt sind:

$R/\overline{W} = L$
$\overline{UDS} = L$
$\overline{LDS} = L$

1.5: Welche Operation wird ausgeführt, wenn folgende Signale gesetzt sind:

$R/\overline{W} = H$
$\overline{UDS} = H$
$\overline{LDS} = H$

1.6: Wie heißen die fünf Signale, die bei einer Datenübertragung im Asynchron-Modus eine Rolle spielen?

1.7: Welche Bedeutung hat das Signal $\overline{BGACK}$?

1.8: Geben Sie die Anzahl der Datenregister an!

1.9: Geben Sie die Anzahl der Adreßregister an!

1.10: Wodurch wird deutlich, daß sich der Prozessor im Supervisor-Zustand befindet?

Lösungen

1.1: Der Adreßbus besitzt 23 Leitungen. Das Bit A0 ist durch $\overline{UDS}$, $\overline{LDS}$ ersetzt worden.

1.2: Ein Wort beginnt immer mit einer geraden Adresse.

1.3: Das höherwertige Byte beginnt mit einer geraden Adresse, das niederwertige mit einer ungeraden.

1.4: Es wird ein Wort geschrieben:

R/$\overline{W}$ = L Schreibvorgang
$\overline{UDS}$= L höherwertiges Byte ist gesetzt
$\overline{LDS}$= L niederwertiges Byte ist gesetzt

1.5: Ein Byte mit einer ungeraden Adresse wird gelesen:

R/$\overline{W}$ = H Lesevorgang
$\overline{UDS}$= H höherwertiges Byte ist nicht gesetzt
$\overline{LDS}$= L niederwertiges Byte ist gesetzt

1.6: Die fünf Signale, die im Asynchron-Modus eine Rolle spielen, heißen: R/$\overline{W}$, $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, $\overline{DTACK}$.

1.7: Wenn das Signal $\overline{BGACK}$ bestätigt worden ist, weiß der Prozessor, daß ein intelligenter Baustein die Steuerung der Busse übernommen hat.

1.8: Der 68000 besitzt 8 Datenregister: D0, .., D7.

1.9: Er besitzt 7 Adreßregister: A0, ..., A6.

1.10: Der Supervisor-Modus wird intern durch Setzen des S-Bits manifestiert und nach außen hin durch den Inhalt der Funktionscodes FC0, FC1 und FC2 signalisiert.

Kapitel 2

Allgemeine Organisation von Datenübertragungen

ÜBERTRAGUNGEN IM ASYNCHRON-MODUS

Dieser Übertragungstyp ermöglicht die Kommunikation zwischen dem 68000 und asynchronen Peripheriegeräten. Dadurch werden die Probleme, die durch die unterschiedlichen Zugriffszeiten der Bausteine entstehen, verringert, und Übertragungsfehler treten selten auf. In diesem Modus findet ein Dialog zwischen dem übergeordneten Baustein oder „Master" (Herr) und einem untergeordneten Baustein oder „Slave" (Sklave) statt. Fragen und Antworten kennzeichnen diesen Dialog. Das Hauptmerkmal dieses Übertragungstyps liegt darin, daß die Zeit in keinerlei Weise beschränkt ist.

In dem Fall, daß die Antwort ausbleibt, wird das System durch eine Alarmuhr (watch-dog) benachrichtigt. Übertragungen im Asynchron-Modus sind kontrollierbar, und bei Auftreten einer Unregelmäßigkeit kann eingegriffen werden: Adreßfehler, nicht vorhandene Adresse usw.

Schließlich erlaubt der Asynchron-Modus auf einfache Weise die Kommunikation mit allen Bausteinen der 68000-Familie wie zum Beispiel dem 68451, einem Speicherbaustein.

LESEZYKLUS

Während des Lesens empfängt der Prozessor Daten aus dem Speicher oder von einem Peripheriegerät.

Er liest immer Bytes.

Wenn der Befehl eine Wortoperation beinhaltet, liest der Prozessor die zwei Bytes über den Bus, in dem Fall sind $\overline{UDS}=0$ und $\overline{LDS}=0$.

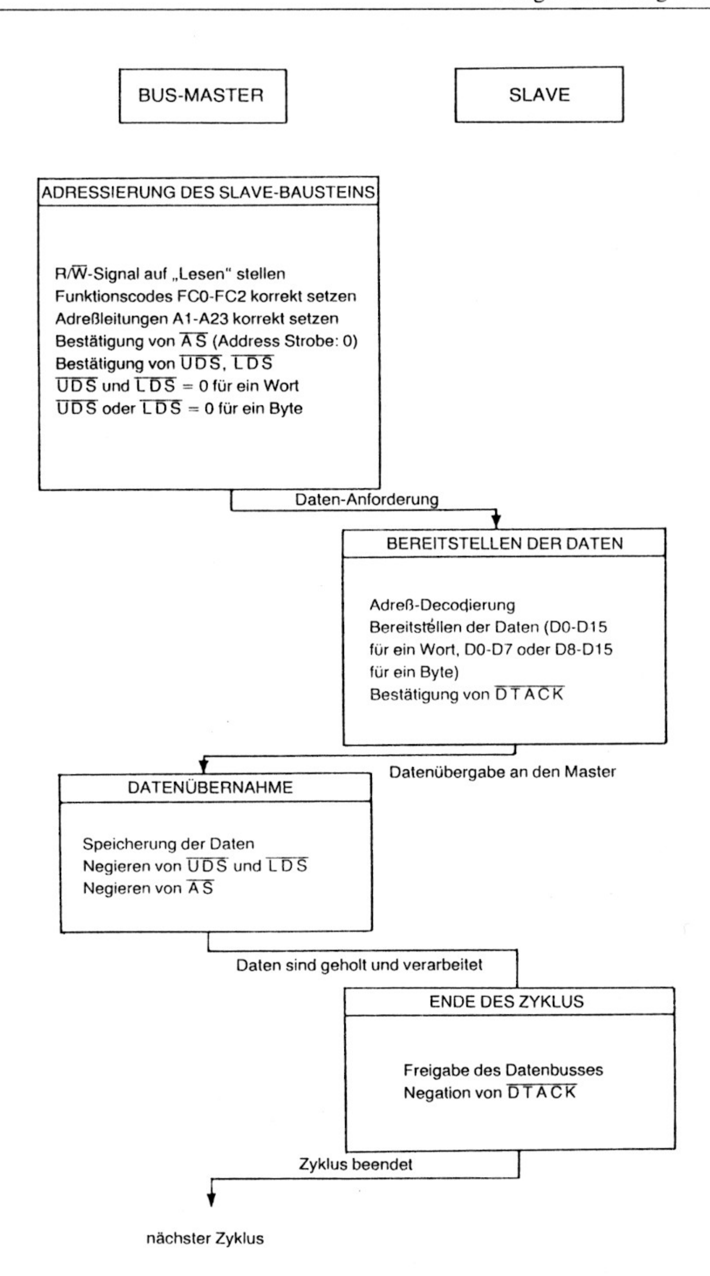


Abb. 2.1: Organisatorischer Ablaufplan eines Lesevorgangs

Enthält der Befehl eine Byteoperation, dann benutzt der Prozessor das interne Bit A0, um zu bestimmen, welches Byte gelesen werden soll:

A0=0 dann sind $\overline{UDS}$=0 und $\overline{LDS}$=1: gerades Byte
A0=1 dann sind $\overline{UDS}$=1 und $\overline{LDS}$=0: ungerades Byte

Um den zeitlichen Ablauf der Signale in einem Lesezyklus besser zu verstehen, betrachten Sie die Diagramme in Abb. 2.1 und 2.2.

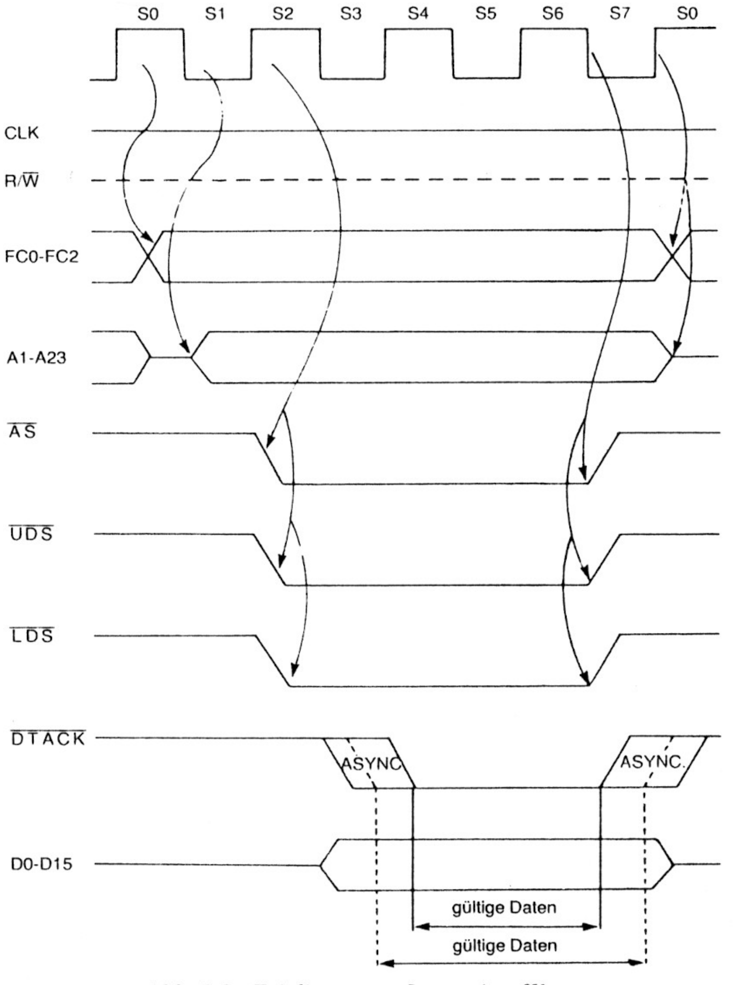


Abb. 2.2: Zeitdiagramm: Lesen eines Wortes

Erläuterung des Zeitdiagramms (Abb. 2.2)

Zustand S0:

- Der Adreßbus ist hochohmig.
- Der Funktionscode wird über FC0, FC1, FC2 angegeben.
- Das Signal R/$\overline{W}$ ist auf High gesetzt, damit der Lesevorgang beginnen kann.

Zustand S1:

- Die Adressen werden auf den Adreßbus gelegt.

Zustand S2:

- $\overline{AS}$ wird durch den Prozessor bestätigt (auf Low gesetzt), um darauf hinzuweisen, daß sich auf dem Adreßbus eine gültige Adresse befindet.
- $\overline{UDS}$, $\overline{LDS}$ werden ebenfalls gesetzt, je nachdem, was gelesen werden soll (Wort oder Byte).

Zustand S3:

- Der untergeordnete Baustein (Slave) wird über den Adreßbus und das Signal $\overline{AS}$ ausgewählt.

Dieser Baustein wertet R/$\overline{W}$, $\overline{UDS}$ und $\overline{LDS}$ aus, um seine Information auf den Bus zu legen. Gleichzeitig bestätigt er $\overline{DTACK}$ (setzt es auf Low), um darauf hinzuweisen, daß die Daten auf dem Bus verfügbar sind. Dieses Signal muß ankommen, bevor der Zyklus S4 beendet ist (beim 68000 L8: mindestens 20 ns vor dem Ende von S4; dies wird durch die „asynchronous setup time“ (asynchrone Vorbereitungszeit) festgelegt. Sonst werden Wartezustände generiert (Swait).

Zustand S4:

- Abtastung von $\overline{DTACK}$.

Zustand S5:

- Interne Synchronisierung von $\overline{DTACK}$.

Zustand S6:

- Speichern der Daten in einem internen Register.

Zustand S7:

– $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$ werden vom Prozessor gelöscht (auf High gesetzt).

Um jegliche Probleme einer schlechten Synchronisation auszuschalten, bleiben alle Adressen gesetzt. Das Signal R/$\overline{W}$ und die Funktionscodes FC0, FC1, FC2 bleiben ebenfalls erhalten, um eine fehlerfreie Datenübertragung zu gewährleisten. Der untergeordnete Baustein gibt anschließend den Datenbus frei und löscht $\overline{DTACK}$, nachdem er die Löschung der Signale $\overline{AS}$, $\overline{LDS}$, $\overline{UDS}$ erkannt hat. Auf diese Weise bleibt der Datenbus gesperrt, und auch das Signal $\overline{DTACK}$ bleibt bis zum Eintreten der Zustände S0, S1 des nächsten Zyklus gelöscht.

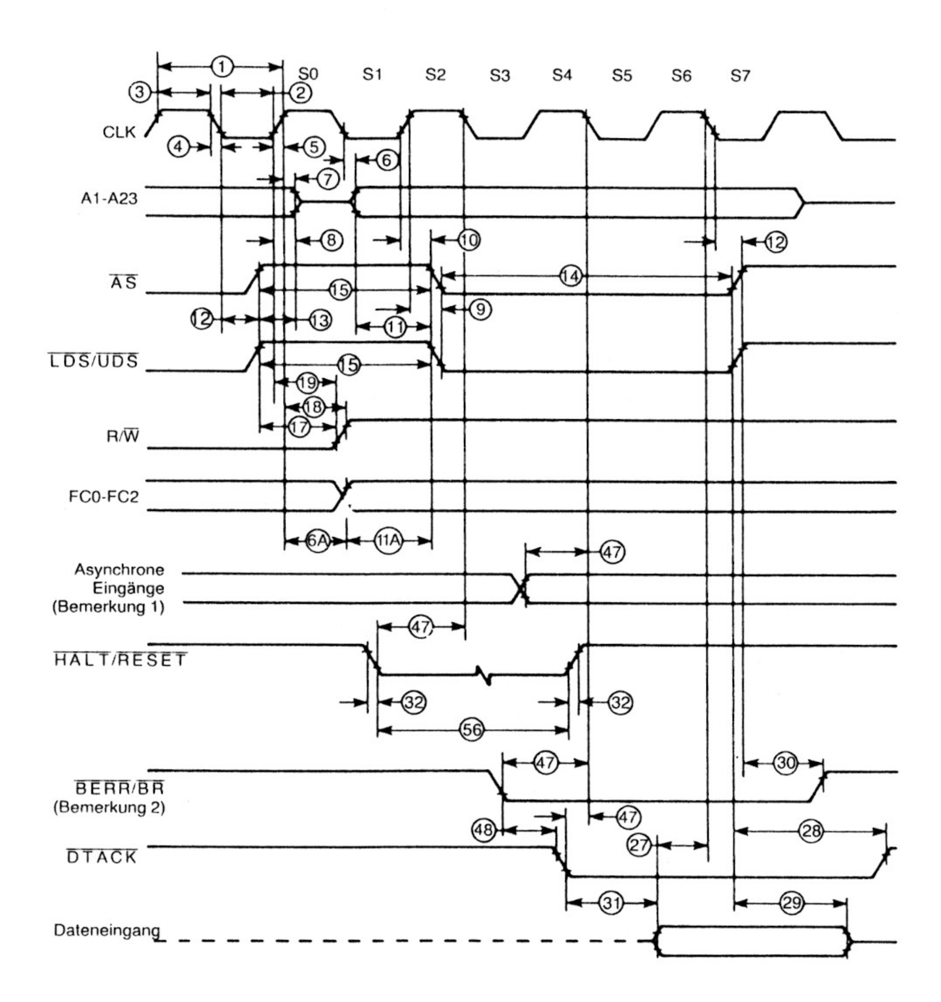
Erzeugen von Wartezuständen Sw:

Wenn das Signal $\overline{DTACK}$ nicht vor dem Zustand S4 gesetzt wird, ersetzt der Prozessor die Zustände S5 und S6 durch Wartezustände Sw (Abb. 2.5). Diese Wartezustände treten paarweise auf. Zwei Sw-Zyklen entsprechen in der Version L8 128 ns. Wenn aber das Signal $\overline{DTACK}$ überhaupt nicht empfangen wird, tritt eine Signaluhr in Kraft, die eine beliebige Unterbrechung oder das Signal $\overline{BERR}$ (Busfehler) erzeugt.

Das genaue Zeitdiagramm eines Lesezyklus und die verschiedenen Zeitabstände sind in Abb. 2.3 und 2.4 dargestellt.

Es ist klar, daß einige Bausteine wie Speicher, Peripheriegeräte etc. nicht von sich aus das Signal $\overline{DTACK}$ erzeugen können. Aus diesem Grund muß eine externe Logik geschaffen werden, die die Zeitintervalle vom Moment der Betätigung der Signale $\overline{UDS}$, $\overline{LDS}$ an zählt, und zwar abhängig von der Zugriffszeit des Bausteins.

Wenn die Übertragung zu Ende ist, werden $\overline{UDS}$ und $\overline{LDS}$ gelöscht, und der Generator von $\overline{DTACK}$ wird wieder auf Null gesetzt. Wenn nach Ablauf einer gewissen Zeit das Signal $\overline{DTACK}$ nicht empfangen wird, meldet die Signaluhr einen Busfehler an den Prozessor. Das Schema der Abb. 2.6 stellt ein Beispiel zur Erzeugung des Signals $\overline{DTACK}$ dar.



BEMERKUNGEN

1. Die Vorbereitungszeit für die asynchronen Eingänge BGACK, IPL0-2 und VPA garantiert ihre korrekte Erkennung bei der nächsten fallenden Flanke des Taktsignals.
2. BR muß zu diesem Zeitpunkt nur auf LOW gehen, damit sichergestellt ist, daß es am Ende des Buszyklus erkannt wird.
3. Den Zeitmessungen liegt ein High-Pegel von 2 Volt und ein Low-Pegel von 0,8 Volt zugrunde, wenn nicht anders angegeben.

Abb. 2.3: Zeitdiagramm eines Lesezyklus

Num.	Characteristic	Symbol	4 MHz		6 MHz		8 MHz		10 MHz		12.5 MHz		Unit
			Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	
1	Clock Period	t_{cyc}	250	500	167	500	125	500	100	500	80	250	ns
2	Clock Width Low	t_{CL}	115	250	75	250	55	250	45	250	35	125	ns
3	Clock Width High	t_{CH}	115	250	75	250	55	250	45	250	35	125	ns
4	Clock Fall Time	t_{Cf}	–	10	–	10	–	10	–	10	–	5	ns
5	Clock Rise Time	t_{Cr}	–	10	–	10	–	10	–	10	–	5	ns
6	Clock Low to Address	t_{CLAV}	–	90	–	80	–	70	–	60	–	55	ns
6A	Clock High to FC Valid	t_{CHFCV}	–	90	–	80	–	70	–	60	–	55	ns
7	Clock High to Address Data High Impedance (Maximum)	t_{CHAZx}	–	120	–	100	–	80	–	70	–	60	ns
8	Clock High to Address/FC Invalid (Minimum)	t_{CHAZn}	0	–	0	–	0	–	0	–	0	–	ns
9[1]	Clock High to $\overline{AS}$, $\overline{DS}$ Low (Maximum)	t_{CHSLx}	–	80	–	70	–	60	–	55	–	55	ns
10	Clock High to $\overline{AS}$, $\overline{DS}$ Low (Minimum)	t_{CHSLn}	0	–	0	–	0	–	0	–	0	–	ns
11[2]	Address to $\overline{AS}$, $\overline{DS}$ (Read) Low/$\overline{AS}$ Write	t_{AVSL}	55	–	35	–	30	–	20	–	0	–	ns
11A[2,7]	FC Valid to $\overline{AS}$, $\overline{DS}$ (Read) Low/$\overline{AS}$ Write	t_{FCVSL}	80	–	70	–	60	–	50	–	40	–	ns
12[1]	Clock Low to $\overline{AS}$, $\overline{DS}$ High	t_{CLSH}	–	90	–	80	–	70	–	55	–	50	ns
13[2]	$\overline{AS}$, $\overline{DS}$ High to Address/FC Invalid	t_{SHAZ}	60	–	40	–	30	–	20	–	10	–	ns
14[2,5]	$\overline{AS}$, $\overline{DS}$ Width Low (Read)/$\overline{AS}$ Write	t_{SL}	535	–	337	–	240	–	195	–	160	–	ns
14A[2]	$\overline{DS}$ Width Low (Write)	t_{DWPW}	285	–	170	–	115	–	95	–	80	–	ns
15[2]	$\overline{AS}$, $\overline{DS}$ Width High	t_{SH}	285	–	180	–	150	–	105	–	65	–	ns
16	Clock High to $\overline{AS}$, $\overline{DS}$ High Impedance	t_{CHSZ}	–	120	–	100	–	80	–	70	–	60	ns
17[2]	$\overline{AS}$, $\overline{DS}$ High to R/$\overline{W}$ High	t_{SHRH}	60	–	50	–	40	–	20	–	10	–	ns
18[1]	Clock High to R/$\overline{W}$ High (Maximum)	t_{CHRHx}	–	90	–	80	–	70	–	60	–	60	ns
19	Clock High to R/$\overline{W}$ High (Minimum)	t_{CHRHn}	0	–	0	–	0	–	0	–	0	–	ns
20[1]	Clock High to R/$\overline{W}$ Low	t_{CHRL}	–	90	–	80	–	70	–	60	–	60	ns
20A[8]	$\overline{AS}$ Low to R/$\overline{W}$ Valid	t_{ASRV}	–	20	–	20	–	20	–	20	–	20	ns
21[2]	Address Valid to R/$\overline{W}$ Low	t_{AVRL}	45	–	25	–	20	–	0	–	0	–	ns
21A[2,7]	FC Valid to R/$\overline{W}$ Low	t_{FCVRL}	80	–	70	–	60	–	50	–	30	–	ns
22[2]	R/$\overline{W}$ Low to $\overline{DS}$ Low (Write)	t_{RLSL}	200	–	140	–	80	–	50	–	30	–	ns
23	Clock Low to Data Out Valid	t_{CLDO}	–	90	–	80	–	70	–	55	–	55	ns
24	Clock High to R/$\overline{W}$, $\overline{VMA}$ High Impedance	t_{CHRZ}	–	120	–	100	–	80	–	70	–	60	ns
25[2]	$\overline{DS}$ High to Data Out Invalid	t_{SHDO}	60	–	40	–	30	–	20	–	15	–	ns
26[2]	Data Out Valid to $\overline{DS}$ Low (Write)	t_{DOSL}	55	–	35	–	30	–	20	–	15	–	ns
27[6]	Data In to Clock Low (Setup Time)	t_{DICL}	30	–	25	–	15	–	10	–	10	–	ns
28[2,5]	$\overline{AS}$, $\overline{DS}$ High to $\overline{DTACK}$ High	t_{SHDAH}	0	490	0	325	0	245	0	190	0	150	ns
29	$\overline{DS}$ High to Data Invalid (Hold Time)	t_{SHDI}	0	–	0	–	0	–	0	–	0	–	ns
30	$\overline{AS}$, $\overline{DS}$ High to $\overline{BERR}$ High	t_{SHBEH}	0	–	0	–	0	–	0	–	0	–	ns
31[2,6]	$\overline{DTACK}$ Low to Data In (Setup Time)	t_{DALDI}	–	180	–	120	–	90	–	65	–	50	ns
32	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	$t_{RHr,f}$	0	200	0	200	0	200	0	200	0	200	ns
33	Clock High to $\overline{BG}$ Low	t_{CHGL}	–	90	–	80	–	70	–	60	–	50	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}	–	90	–	80	–	70	–	60	–	50	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clk Per.
36	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clk Per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk Per.

Abb. 2.4: Lese- und Schreibzyklus (Spezifikationen)

Num.	Characteristic	Symbol	4 MHz		6 MHz		8 MHz		10 MHz		12.5 MHz		Unit
			Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	
37A	$\overline{BGACK}$ Low to $\overline{BR}$ High (to Prevent Rearbitration)	t_{BGKBR}	*)		25		20	–	20		20	–	ns
38	$\overline{BG}$ Low to Bus High Impedance (with $\overline{AS}$ High)	t_{GLZ}	–	120	–	100	–	80	–	70	–	60	ns
39	$\overline{BG}$ Width High	t_{GH}	1.5	–	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.
40	Clock Low to $\overline{VMA}$ Low	t_{CLVML}	–	90	–	80	–	70	–	70	–	70	ns
41	Clock Low to E Transition	t_{CLC}	–	100	–	85	–	70	–	55	–	45	ns
42	E Output Rise and Fall Time	$t_{Er,f}$	–	25	–	25	–	25	–	25	–	25	ns
43	$\overline{VMA}$ Low to E High	t_{VMLEH}	325	–	240	–	200	–	150	–	90	–	ns
44	$\overline{AS}$, $\overline{DS}$ High to $\overline{VPA}$ High	t_{SHVPH}	0	240	0	160	0	120	0	90	0	70	ns
45	E Low to Address $\overline{VMA}$ FC Invalid	t_{ELAI}	55	–	35	–	30	–	10	–	10	–	ns
46	$\overline{BGACK}$ Width	t_{BGL}	1.5	–	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.
47[6]	Asynchronous Input Setup Time	t_{ASI}	30	–	25	–	20	–	20	–	20	–	ns
48[3]	$\overline{BERR}$ Low to $\overline{DTACK}$ Low	t_{BELDAL}	30	–	25	–	20	–	20	–	20	–	ns
49	E Low to $\overline{AS}$, $\overline{DS}$ Invalid	t_{ELSI}	-80	–	-80	–	-80	–	-80	–	-80	–	ns
50	E Width High	t_{EH}	900	–	600	–	450	–	350	–	280	–	ns
51	E Width Low	t_{EL}	1400	–	900	–	700	–	550	–	440	–	ns
52	E Extended Rise Time	t_{CIEHX}		80	–	80	–	80	–	80	–	80	ns
53	Data Hold from Clock High	t_{CHDO}	0	–	0	–	0	–	0	–	0	–	ns
54	Data Hold from E Low (Write)	t_{ELDOZ}	60	–	40	–	30	–	20	–	15	–	ns
55	R/$\overline{W}$ to Data Bus Impedance Change	t_{RLDO}	55	–	35	–	30	–	20	–	10	–	ns
56[4]	$\overline{HALT}$/$\overline{RESET}$ Pulse Width	t_{HRPW}	10	–	10	–	10	–	10	–	10	–	Clk. Per.

Abb. 2.4: Lese- und Schreibzyklus (Fortsetzung)

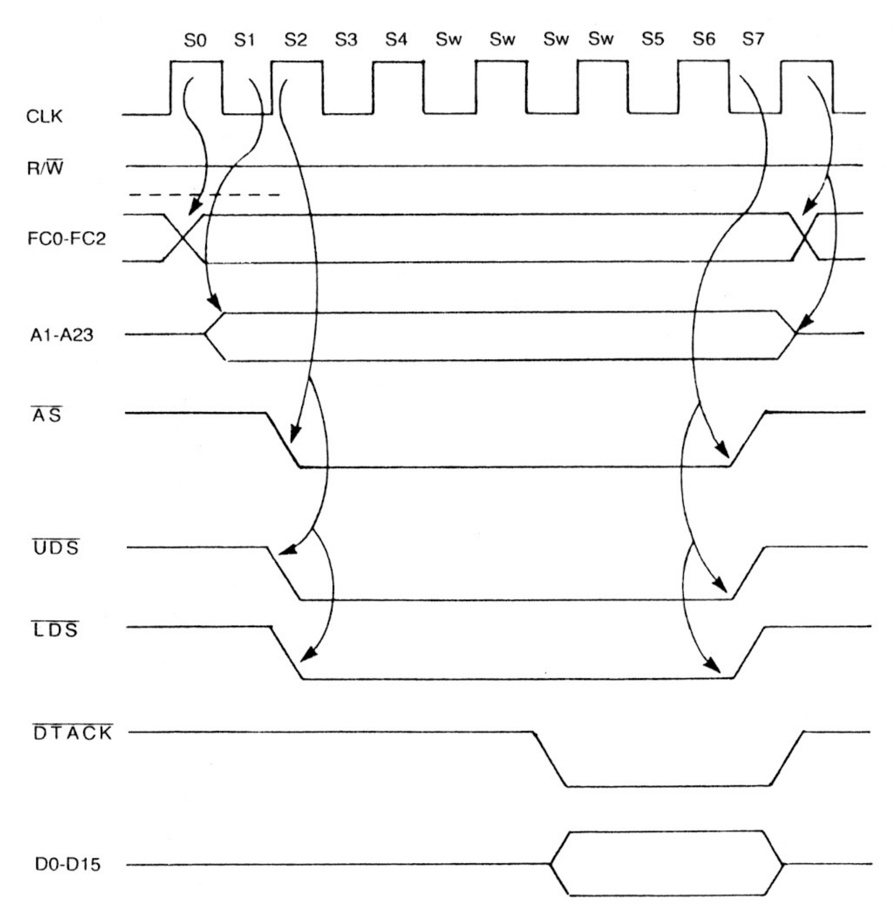


Abb. 2.5: Zeitdiagramm eines Lesezyklus mit Erzeugen von Wartezuständen

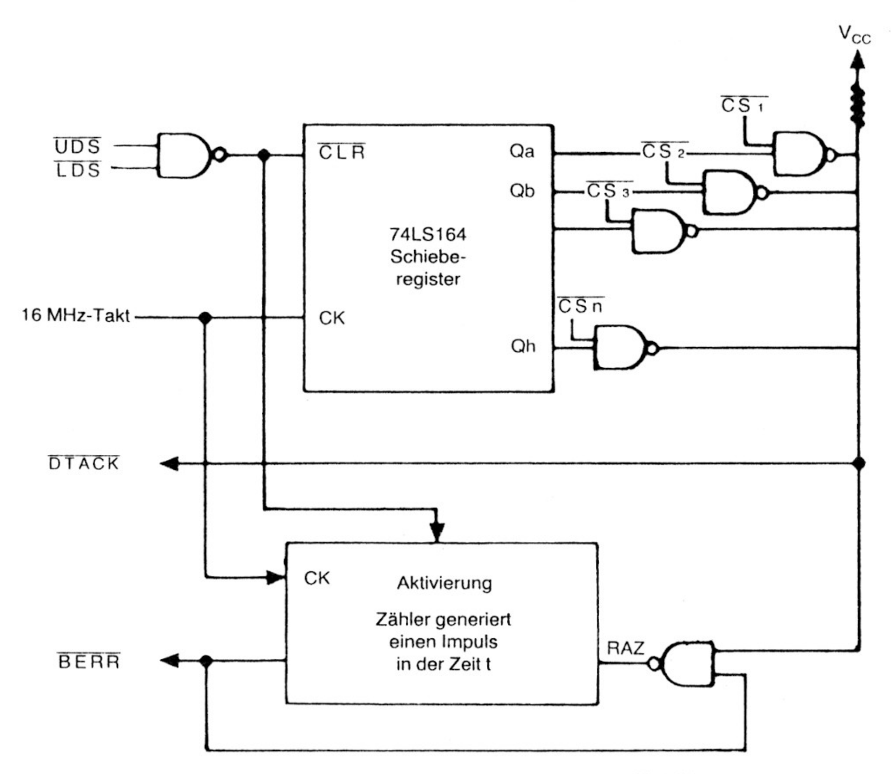


Abb. 2.6: Beispiel zur Erzeugung des Signals $\overline{DTACK}$

SCHREIBZYKLUS

Während eines Schreibzyklus sendet der Prozessor Daten zu einem Speicher oder Peripheriegerät. In jedem Fall schreibt der Prozessor Bytes. Wenn der Befehl eine Wort-Operation enthält, schreibt der Prozessor zwei Bytes. Beinhaltet der Befehl eine Byte-Operation, benutzt der Prozessor das interne Bit A0, um festzustellen, in welche Adresse er schreiben soll und welche entsprechenden Signale er setzen soll.

Wenn A0=0, dann setze $\overline{UDS}$=0 und $\overline{LDS}$=1: Schreiben eines Bytes an eine gerade Adresse.

Wenn A0=1, dann setze $\overline{UDS}$=1 und $\overline{LDS}$=0: Schreiben eines Bytes an eine ungerade Adresse.

Wir wollen nun einen Schreibzyklus anhand der Schemazeichnung Abb. 2.7 und des Zeitdiagramms Abb. 2.8 untersuchen.

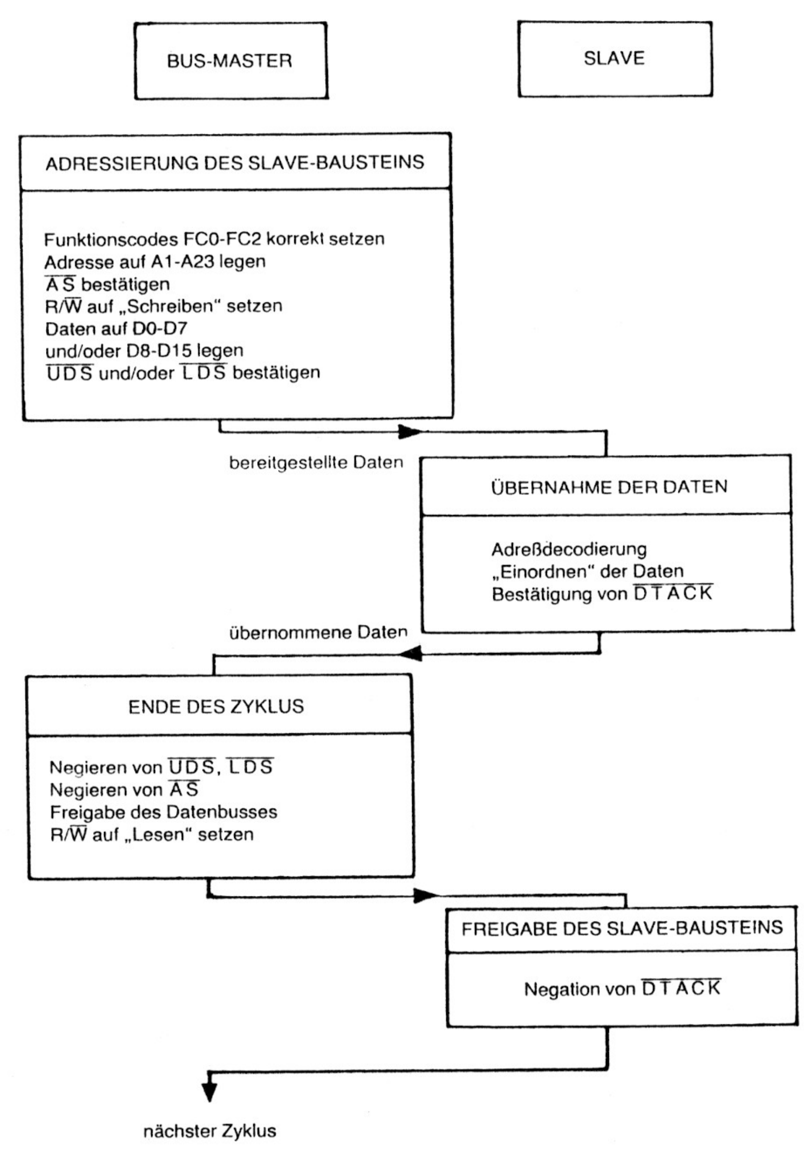


Abb. 2.7: Schemazeichnung eines Lesezyklus

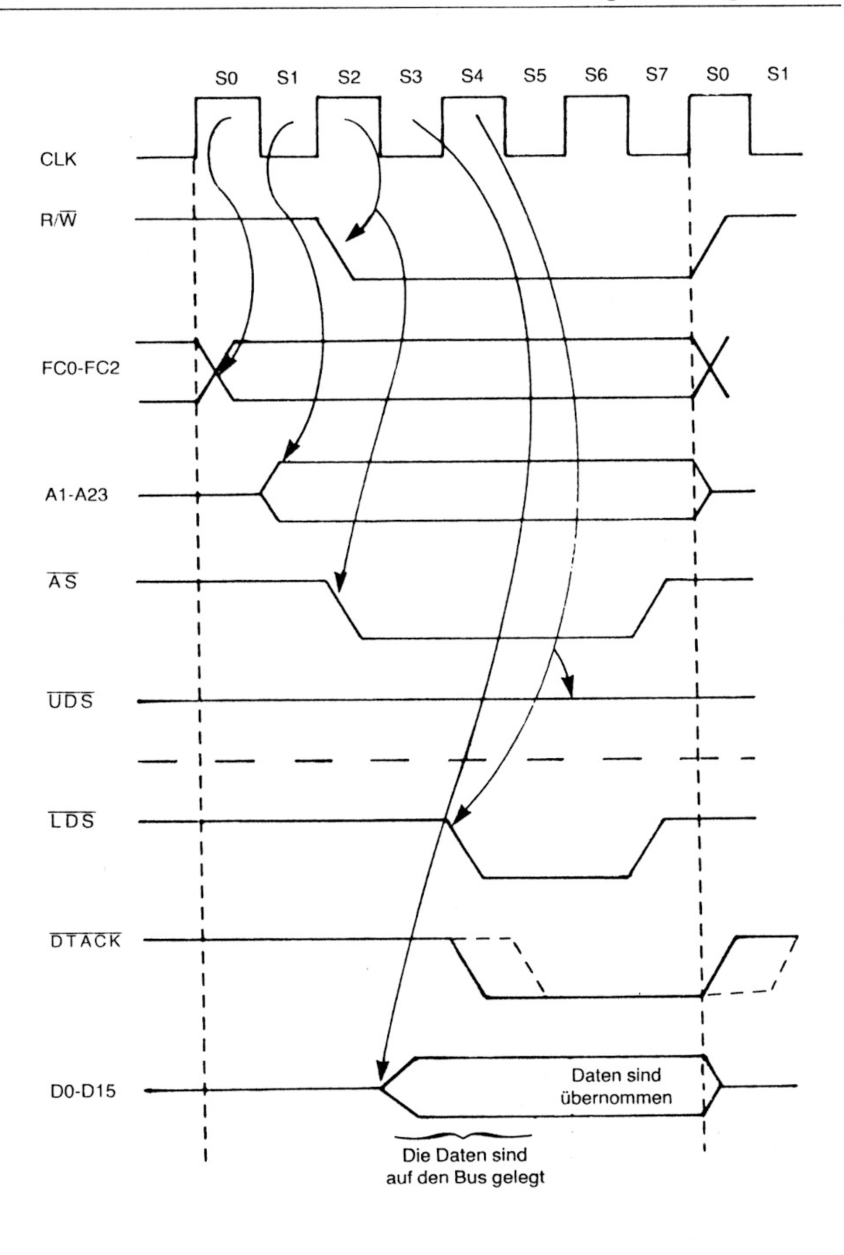


Abb. 2.8: Zeitdiagramm eines Lesezyklus

Erläuterung des Zeitdiagramms (Abb. 2.8)

Zustand S0:

- Der Adreßbus ist hochohmig.
- Die Funktionscodes werden über FC0, FC1, FC2 ausgegeben.

Zustand S1:

- Die Adressen werden auf den Adreßbus gelegt.

Zustand S2:

- $\overline{AS}$ wird bestätigt, um darauf hinzuweisen, daß sich auf dem Adreßbus eine gültige Adresse befindet.
- Der Peripheriebaustein oder der Speicher wird über den Adreßbus und $\overline{AS}$ angewählt.
- Das Signal R/$\overline{W}$ ist für den Schreibvorgang gesetzt (low).

Zustand S3:

- Der Prozessor legt die Daten auf den Bus.

Zustand S4:

- Die Signale $\overline{UDS}$ und $\overline{LDS}$ werden gesetzt: $\overline{UDS}$ und/oder $\overline{LDS}=0$.

Der angewählte Baustein verwendet das Signal R/$\overline{W}$ sowie $\overline{UDS}$ und $\overline{LDS}$, um dem Datenbus die Information zu entnehmen. Nachdem er die Daten gespeichert hat, bestätigt er das Signal $\overline{DTACK}$. Wie beim Lesezyklus muß $\overline{DTACK}$ vor dem Ende des Zustands S4 empfangen werden, sonst wird ein Wartezustand erzeugt.

Zustand S5:

- Interne Synchronisierung des Signals $\overline{DTACK}$.

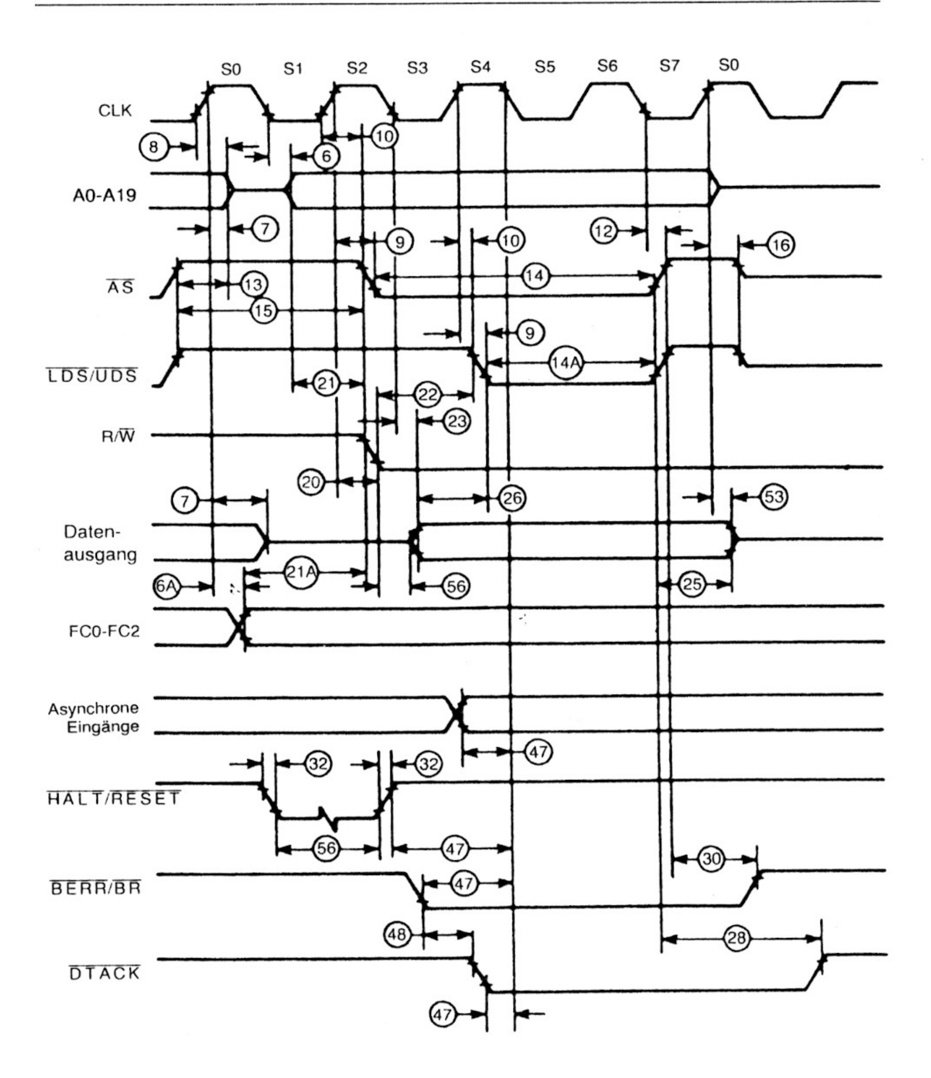
Zustand S6:

- Stabiler Zustand.

Zustand S7:

- Die Signale $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$ werden gelöscht (auf High gesetzt).

Die Werte auf dem Daten- und Adreßbus bleiben gültig. Das Signal R/$\overline{W}$ bleibt ebenfalls aufrechterhalten, um eine fehlerfreie Datenübertragung



BEMERKUNGEN

1. Bei Zeitmessungen wird ein Low-Pegel von 0,8 Volt und ein High-Pegel von 2,0 Volt vorausgesetzt, wenn nicht ausdrücklich anders erwähnt.
2. Es ist möglich, daß R/$\overline{W}$ erst nach $\overline{AS}$ gültig wird, obwohl beide von der steigenden Flanke von S2 gesteuert werden.

Abb. 2.9: Zeitdiagramm eines Schreibzyklus

zu gewährleisten. Der untergeordnete Baustein (Slave) betrachtet $\overline{DTACK}$ so lange als gesetzt, bis er die Löschung des Signals $\overline{AS}$ oder $\overline{UDS}$, $\overline{LDS}$ wahrnimmt. Anschließend löscht er dann nach einem gewissen Zeitintervall das Signal $\overline{DTACK}$. Zum Schluß gibt der Prozessor am Ende von S7 oder während des Zustands S0 des darauffolgenden Zyklus die Busse frei. Das genaue Zeitdiagramm eines Schreibzyklus ist in Abb. 2.9 zu sehen.

LESE-/ÄNDERUNGS-/SCHREIBZYKLUS

Der Lese-/Änderungs-/Schreibzyklus realisiert einen Lesevorgang, ändert die Daten in der arithmetisch-logischen Einheit (ALU) und schreibt die Daten an die gleiche Adresse, an der sie gelesen wurden. Das Signal $\overline{AS}$ bleibt während der 3 Phasen Lesen, Ändern und Schreiben gültig, so daß eine Unterbrechung dieses Zyklus unmöglich ist.

Der Befehl TAS (Test And Set), Testen und Setzen des höchstwertigen Bits des Bytes, ist der einzige, der diesen Zyklustyp verwendet. Er ermöglicht das Zugreifen auf gemeinsame Datenfelder in Multiprozessorsystemen.

Der Befehl TAS ist eine Byte-Operation, was eine Bestätigung dafür ist, daß Lese-/Änderungs-/Schreibzyklen stets Byte-Operationen sind. Da während der Datenübertragung das Signal $\overline{AS}$ gesetzt ist, kann eine andere Busanfrage erst nach Beendigung des TAS-Befehls berücksichtigt werden.

Dieser Zyklustyp dauert mindestens 10 Takte (20 Zustände). Die Bedeutung dieses Zyklus wird Ihnen nach näherer Betrachtung des Befehls $\overline{TAS}$ in der Befehlsübersicht noch klarer werden. Die Schemazeichnung des Lese-/Änderungs-/Schreibzyklus ist in Abb. 2.10 dargestellt und das zugehörige Zeitdiagramm in Abb. 2.11.

Erläuterung des Zeitdiagramms (Abb. 2.11)

Bis zum Zustand S6 wird ein normaler Lesezyklus ausgeführt.

Zustand S7:

- $\overline{UDS}$ und $\overline{LDS}$ werden gelöscht.
- Der Adreßbus, $\overline{AS}$, $R/\overline{W}$ und die Funktionscodes bleiben bei der Vorbereitung des Schreibvorgangs unverändert.
- $\overline{DTACK}$ wird vom untergeordneten Baustein nach dem Löschen von $\overline{UDS}$ und $\overline{LDS}$ ebenfalls gelöscht.

Zustand S8:

- Die interne Änderung der Daten kann also durchgeführt werden.
- Keine Änderung der Steuersignale.
- Das Signal R/$\overline{W}$ bleibt bis zum Eintreten des Zustands S14 auf High gesetzt, um Buskonflikte mit dem vorhergegangenen Lesevorgang zu vermeiden.

Zustände S9 bis S13:

- Keine Änderung der Steuersignale.

Zustand S14:

- Das Signal R/$\overline{W}$ wird für den Schreibvorgang gesetzt (low). Dieser Zustand ist gleichbedeutend mit dem Zustand S2 eines normalen Schreibvorgangs.

Zustand S15:

- Die Daten werden auf den Bus gelegt. Der Zustand ist äquivalent zum Zustand S3 des normalen Schreibvorgangs.

Zustand S16:

- $\overline{UDS}$ und $\overline{LDS}$ werden bestätigt, um anzuzeigen, daß der Datenbus in einem stabilen Zustand ist.
- Das Signal $\overline{DTACK}$ wird erwartet, das vor Beendigung des Zustands S16 empfangen werden muß (sonst wird Sw erzeugt). Dieser Zustand ist mit Zustand S4 des normalen Schreibvorgangs zu vergleichen.

Zustände S17, S18:

- Gleichbedeutend mit S5, S6.

Zustand S19:

- Gleichbedeutend mit S7.

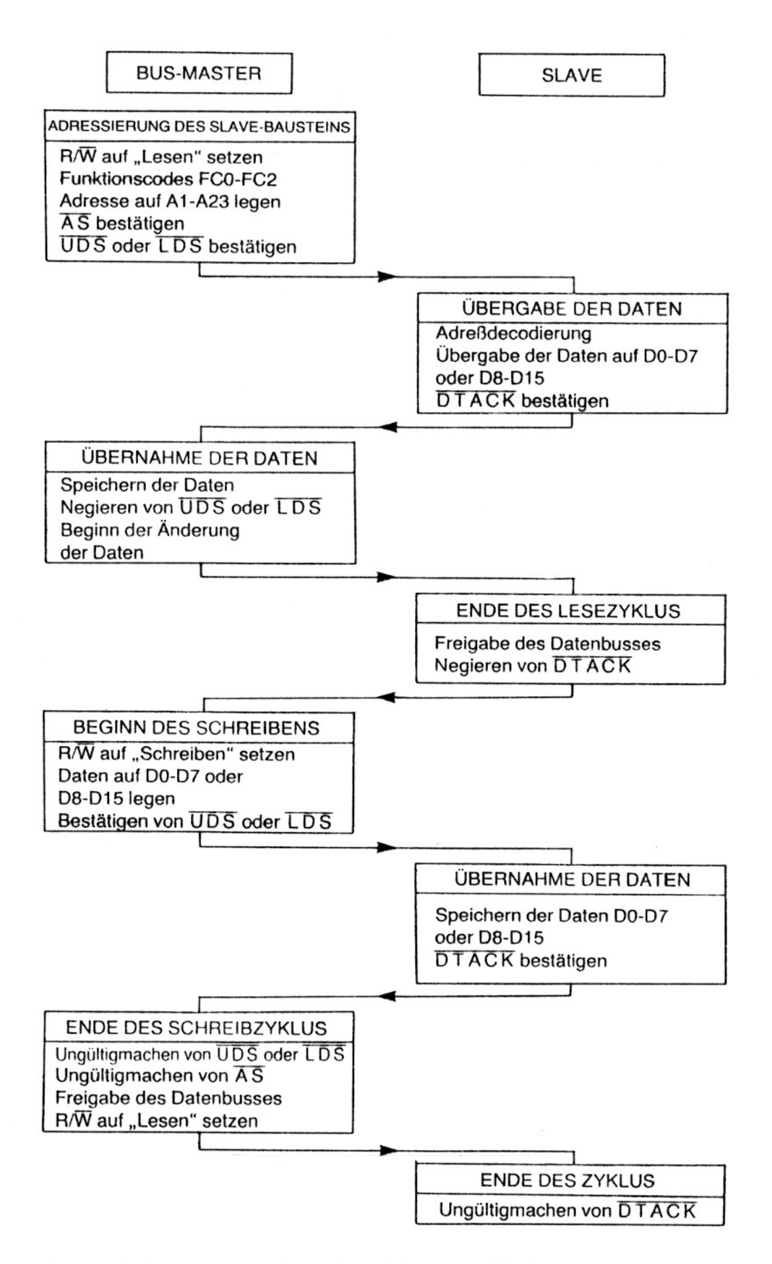


Abb. 2.10: Organisatorischer Ablauf des Lese-/Änderungs-/Schreibzyklus

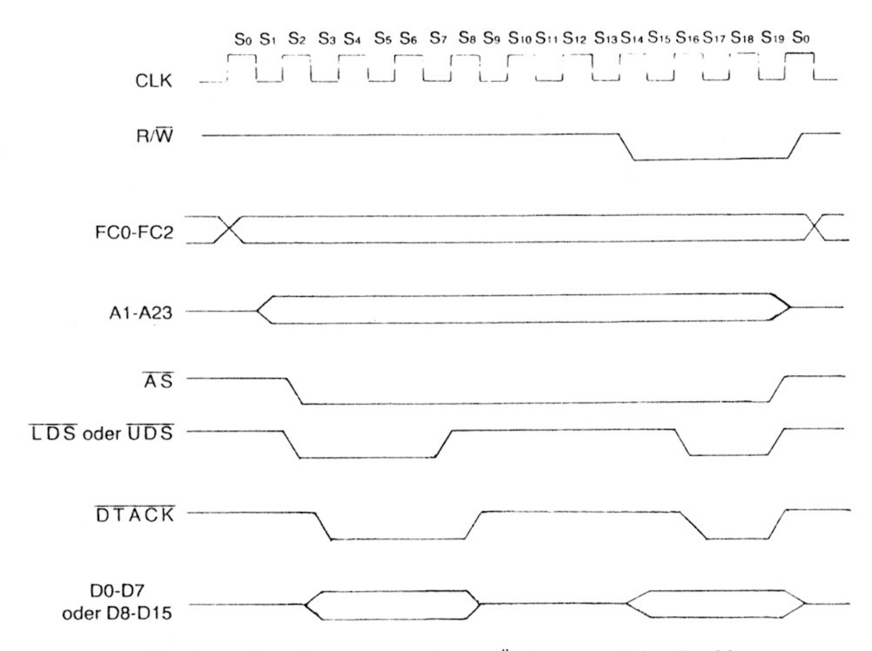


Abb. 2.11: Zeitdiagramm des Lese-/Änderungs-/Schreibzyklus

ZYKLUSSTÖRUNGEN

In einer Busarchitektur, die aufgrund eines Dialogsystems mit einem externen Baustein funktioniert, kann es vorkommen, daß das Frage-Antwortspiel nicht zustandekommt. Es ist dann notwendig, daß eine externe Logik eingeschaltet wird, eine Signaluhr, die im Notfall einen Fehler auf dem Bus meldet: $\overline{BERR}$.

$\overline{BERR}$ (Bus ERRor) tritt immer auf, wenn es bei einer Datenübertragung zu einer Anomalie kommt z. B.:

- Baustein antwortet nicht
- Baustein nicht vorhanden
- Baustein für Fehlerwahrnehmung und -berichtigung meldet sich (68653)
- Seite nicht vorhanden (bei virtueller Adressierung)

Wenn der Prozessor das Signal $\overline{BERR}$ erhält, hat er zwei Möglichkeiten:

1. Einleitung einer Ausnahmeprozedur: Ausführung des Programms, das das Auftreten eines Busfehlers behandelt.
2. Den Buszyklus von neuem aktivieren.

Die Ausnahmeprozedur

Ist das Signal $\overline{BERR}$ bestätigt worden, wird der ablaufende Buszyklus unterbrochen. Adreß- und Datenbus bleiben so lange hochohmig, wie das Signal $\overline{BERR}$ gesetzt bleibt. Wenn $\overline{BERR}$ gelöscht wird, leitet der Prozessor seine „Rettungsprozedur" ein, die zunächst gewisse Informationen auf den Stapel legt.

Die Ausnahmeprozedur für einen Busfehler ist fast die gleiche wie die einer autovektoriellen Unterbrechung (Unterbrechung, deren Vektornummer automatisch vom Prozessor errechnet wird).

Diese Prozedur arbeitet wie folgt:

- Retten des Programmzeigers (PC) und des Statusregisters (SR) auf den Stapel.
- Retten der Fehlerbezeichnung, um den Fehler zu bestimmen (dies ist eine zusätzliche Aktion im Vergleich zur Unterbrechung).
- Ablesen der Adresse des Fehlerprogramms auf dem Bus, die sich in der Ausnahmevektortabelle an der Adresse des Signals $\overline{BERR}$ ($000008) befindet.
- Ausführen des Programms, das dieser Ausnahme entspricht.

Eine genauere Untersuchung von $\overline{BERR}$ wird im Kapitel über die Ausnahmen durchgeführt.

Die Reaktionen der Steuersignale im Falle des Auftretens von $\overline{BERR}$ sind im Zeitdiagramm der Abb. 2.12 dargestellt.

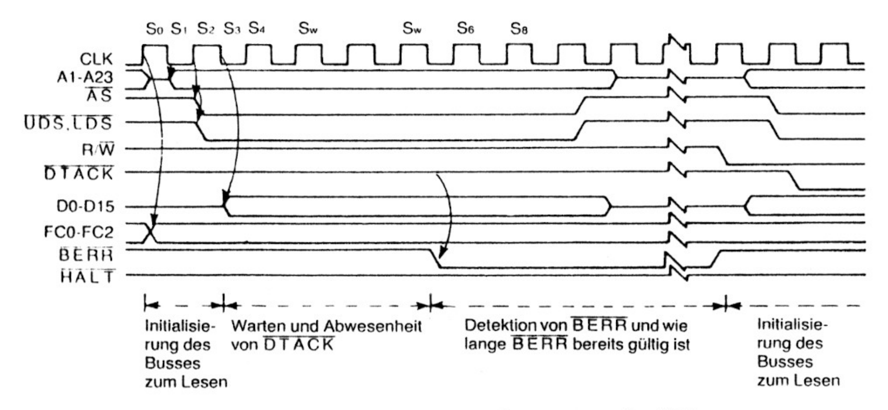


Abb. 2.12: Zeitdiagramm bei Auftreten eines Busfehlers

Wiederaufnahme des Zyklus auf dem Bus (Re-run)

Erhält der Prozessor während des Buszyklus das Signal $\overline{BERR}$ und ist $\overline{HALT}$ durch einen externen Baustein aktiviert worden, dann beginnt der Prozessor noch einmal den gleichen Zyklus.

Zunächst jedoch unterbricht er beim Empfang dieser beiden Signale den gerade ablaufenden Zyklus und versetzt den Adreß- und den Datenbus in den hochohmigen Zustand. Die Signale $\overline{AS}$, $\overline{UDS}$ und $\overline{LDS}$ werden inaktiv. Danach bleibt der Prozessor inaktiv, bis das $\overline{HALT}$-Signal von dem externen Baustein auf inaktiv gesetzt wird.

Nun startet der Prozessor den Buszyklus von neuem, indem er die gleichen Daten (bei einer Schreiboperation) und die gleichen Steuersignale verwendet. Das Signal $\overline{BERR}$ muß mindestens einen Takt vor dem Löschen des $\overline{HALT}$-Signals deaktiviert worden sein.

Bemerkung: Der Prozessor führt kein Re-run während eines Lese-/Änderungs-/Schreibzyklus durch. In der Tat muß bei diesem Operationstyp der Schreibvorgang ausgeführt werden, ohne daß das Signal $\overline{AS}$ seit dem zugehörigen Lesevorgang geändert wurde. Dies ist natürlich nicht möglich, wenn der Prozessor eine Wiederaufnahme des Zyklus durchführt.

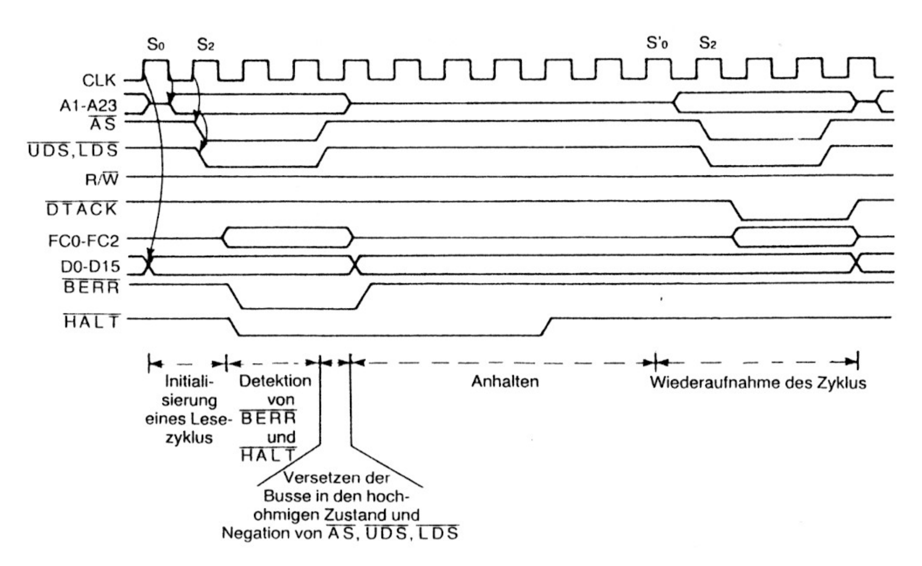


Abb. 2.13: Zeitdiagramm eines Wiederaufnahmezyklus (Re-run)

Der MC68000 erzeugt also einen Busfehler, wenn die Signale $\overline{\text{BERR}}$ und $\overline{\text{HALT}}$ während eines Lese-/Änderungs-/Schreibzyklus gesetzt werden.

Mit Hilfe des Zeitdiagramms in Abb. 2.13 untersuchen wir eine Wiederaufnahme.

Es muß erwähnt werden, daß das Signal $\overline{\text{HALT}}$ ebenfalls verwendet wird, um den „Einzelschrittmodus" zu generieren, der bei der Fehlerbeseitigung in Programmen behilflich sein kann, und daß zwei aufeinanderfolgende Busfehler einen „doppelten Busfehler" ergeben.

Alle diese Fälle werden später in diesem Kapitel und im Kapitel über Ausnahmen besprochen.

Bevor wir die Betrachtungen über Zyklusstörungen abschließen, fassen wir noch einmal die möglichen Aktionen der Signale $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$ und $\overline{\text{HALT}}$ mit Hilfe der Tabellen in Abb. 2.14 und 2.15 zusammen.

AKTION	SIGNALE
Normales Zyklusende und Übergang zum folgenden Zyklus	$\overline{\text{DTACK}}$ kommt als erstes Signal an
Normales Zyklusende und Stillstand; danach Neustart gemäß $\overline{\text{HALT}}$	$\overline{\text{HALT}}$ kommt vor oder zur gleichen Zeit wie $\overline{\text{DTACK}}$ an
Zyklus unterbrochen und Verarbeitung der Ausnahme – Busfehler	$\overline{\text{BERR}}$ bestätigt vor oder zur gleichen Zeit wie $\overline{\text{DTACK}}$ und $\overline{\text{BERR}}$ verneint zur gleichen Zeit oder nach der Negation von $\overline{\text{DTACK}}$
Zyklus unterbrochen und neu gestartet	$\overline{\text{HALT}}$ und $\overline{\text{BERR}}$ bestätigt zur gleichen Zeit oder vor $\overline{\text{DTACK}}$ $\overline{\text{HALT}}$ verneint wenigstens einen Zyklus nach $\overline{\text{BERR}}$

Abb. 2.14: Die verschiedenen Möglichkeiten, einen Zyklus zu beenden

Bemerkung: $\overline{\text{BERR}}$, $\overline{\text{DTACK}}$ $\overline{\text{HALT}}$ müssen an der steigenden Flanke des Eingangstaktes bestätigt oder verneint werden, damit die Signale im gleichen Zustand stabil sind.

Die Art und Weise, wie diese Signale verneint werden, ist ebenfalls sehr wichtig und wird in der Tabelle der Abb. 2.16 dargestellt.

FALL-Nr.	SIGNALE	BESTÄTIGT AN DER GRENZE DES ZUSTANDS N	N+2	ERGEBNISSE
1	$\overline{DTACK}$	A	BA	normales Zyklusende
	$\overline{BERR}$	NA	X	
	$\overline{HALT}$	NA	X	
2	$\overline{DTACK}$	A	BA	normales Zyklusende
	$\overline{BERR}$	NA	X	Stillstand und Wiederbeginn
	$\overline{HALT}$	A	BA	nach Negation von $\overline{HALT}$
3	$\overline{DTACK}$	NA	A	normales Zyklusende
	$\overline{BERR}$	NA	NA	Stillstand und Wiederbeginn
	$\overline{HALT}$	A	BA	nach Negation von $\overline{HALT}$
4	$\overline{DTACK}$	X	X	Zyklus unterbrochen und
	$\overline{BERR}$	A	BA	Verarbeitung der
	$\overline{HALT}$	NA	NA	Ausnahme $\overline{BERR}$
5	$\overline{DTACK}$	X	X	Zyklus unterbrochen und
	$\overline{BERR}$	A	BA	Neustart des Zyklus
	$\overline{HALT}$	NA	A	
6	$\overline{DTACK}$	X	X	Zyklus unterbrochen und
	$\overline{BERR}$	A	BA	Neustart des Zyklus
	$\overline{HALT}$	A	BA	
7	$\overline{DTACK}$	NA	X	Zyklus unterbrochen und
	$\overline{BERR}$	NA	A	Neustart, wenn $\overline{HALT}$
	$\overline{HALT}$	A	BA	verneint ist

A = aktiv NA = nicht aktiv BA = bleibt aktiv X = gleichgültig

Abb. 2.15: Tabelle der Kombinationen der Signale $\overline{DTACK}$, $\overline{BERR}$, $\overline{HALT}$

Beispiele für Zyklusunterbrechungen

Beispiel A:

Ein Decodier-Baustein sendet ein Signal, wenn ein nicht vorhandenes Feld angesprochen wird. In dieser Situation kann er unter anderem $\overline{BERR}$ und $\overline{DTACK}$ gleichzeitig aktivieren, um eine Ausnahmeprozedur „Busfehler" zu erzeugen.

SITUATIONEN VERURSACHT DURCH DIE SIGNALBE-STÄTIGUNGEN	SIGNALE	NEGATION AN DER STEIGENDEN FLANKE DES ZUSTANDS		KONSEQUENZ FÜR DEN NÄCHSTEN ZYKLUS
		N	N+2	
Normal	$\overline{BERR}$ $\overline{HALT}$	N N	– N	kann den Buszyklus verlängern
Normal	$\overline{BERR}$ $\overline{HALT}$	– N	N –	Wenn der Buszyklus begonnen ist, gibt es einen Busfehler
Busfehler	$\overline{BERR}$ $\overline{HALT}$	N N	N N	Ausnahme $\overline{BERR}$
Neustart	$\overline{BERR}$ $\overline{HALT}$	N N	N –	illegal, Verzweigung zum Vektor Nr. 0
Neustart	$\overline{BERR}$ $\overline{HALT}$	N –	– N	Neustart des Buszyklus

N: Negation der Signale im dargestellten Zustand

Abb. 2.16: Tabelle der möglichen $\overline{BERR}$, $\overline{HALT}$

Beispiel B:

Ein Warnsystem, das fehlerhafte Daten in einem Speicherplatz feststellt, sendet ein Signal. Das System wartet, bis die Daten überprüft worden sind, bevor $\overline{DTACK}$ aktiviert wird:

- sind die Daten fehlerfrei, wird $\overline{DTACK}$ aktiviert;
- sind die Daten fehlerhaft, werden gleichzeitig $\overline{BERR}$ und $\overline{HALT}$ aktiviert, so daß der Zyklus von neuem beginnt.

Beispiel C:

Wir gehen wieder von demselben System aus, das wartet, bis die Daten überprüft worden sind, bevor $\overline{DTACK}$ aktiviert wird:

- sind die Daten fehlerfrei, wird $\overline{DTACK}$ aktiviert;
- sind die Daten fehlerhaft, werden $\overline{BERR}$ und $\overline{DTACK}$ aktiviert, um eine Ausnahmeprozedur „Busfehler" einzuleiten.

Bevor wir zur Aufzählung der unterschiedlichen Funktionen des Signals $\overline{HALT}$ übergehen, wollen wir noch einmal in einer Tabelle das Zusammenspiel der Signale $\overline{BERR}$ und $\overline{HALT}$ verdeutlichen.

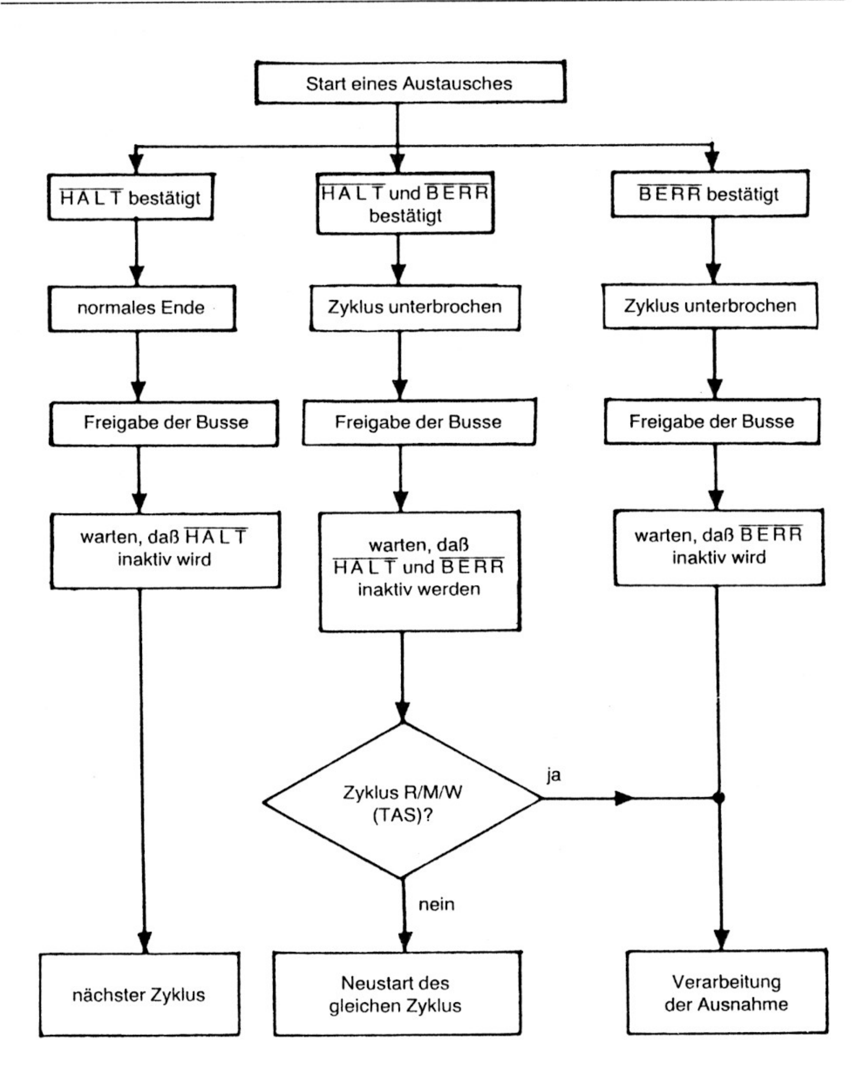


Abb. 2.17: Zusammenspiel der Signale $\overline{BERR}$ und $\overline{HALT}$

WEITERE FUNKTIONEN DES SIGNALS $\overline{HALT}$

Zunächst einmal rufen wir uns in Erinnerung, daß der Anschluß $\overline{HALT}$ bidirektional ist. Wenn wir ihn als Eingang betrachten, können die folgenden Fälle auftreten:

- $\overline{HALT}$ zusammen mit $\overline{BERR}$: bewirkt die Wiederholung des Buszyklus.
- $\overline{HALT}$ zusammen mit $\overline{RESET}$: bewirkt den Neustart des Systems.
- $\overline{HALT}$ alleine: bewirkt den Stillstand des Prozessors.

$\overline{HALT}$ alleine bewirkt den Stillstand des Prozessors, nachdem der laufende Buszyklus beendet ist; die Steuersignale sind in diesem Fall inaktiv ($\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, R/$\overline{W}$).

Die Tristate-Leitungen sind hochohmig. Dennoch werden die Bus-Anforderungen berücksichtigt; die Signale $\overline{BR}$, $\overline{BG}$ und $\overline{BGACK}$ bleiben gültig.

Wenn man diesen Anschlußstift ($\overline{HALT}$) als Ausgang betrachtet, wird er aktiviert, wenn der Prozessor nach einem Doppelbusfehler zum Stillstand kommt, damit die externen Schaltungen auf diesen Zustand aufmerksam werden. RESET wird den Mikroprozessor von neuem starten.

Der Einzelschritt-Modus

Die Verwendung des Signals $\overline{HALT}$ ermöglicht den Ablauf des Buszyklus im Einzelschritt-Modus.

Der Prozessor führt einen Buszyklus aus, kommt zum Stillstand, bearbeitet dann den nächsten Buszyklus usw.

Das Prinzip des Einzelschritt-Modus wird mit Hilfe der Abb. 2.18 verdeutlicht.

Bei diesem Schema fällt auf, daß die Leitung, die mit normal/Einzelschritt beschriftet ist, beim normalen Ablauf auf 0 und im Einzelschritt-Modus auf 1 gesetzt ist. Auf diese Weise wird im Normalmodus, unabhängig vom zweiten Teil der Logik, kein einziges Signal $\overline{HALT}$ erzeugt. Im Einzelschritt-Modus ist dies vom Signal $\overline{AS}$ und dem Zustand des Schalters S2 abhängig. Sobald $\overline{AS}$ aktiv ist, besitzt der Ausgang $\overline{Q}$ des Flip-Flops den Wert 1, und das Signal $\overline{HALT}$ geht auf Low (aktiv). Wenn sich S2 im Wartezustand befindet, verändert sich der Zustand des Flip-Flops nicht, der Prozessor bleibt gestoppt. Geht S2 jedoch in den Einzelschritt-Modus über, wird der Ausgang $\overline{Q}$ auf 1 gesetzt, und das Signal $\overline{HALT}$ ist von neuem aktiv.

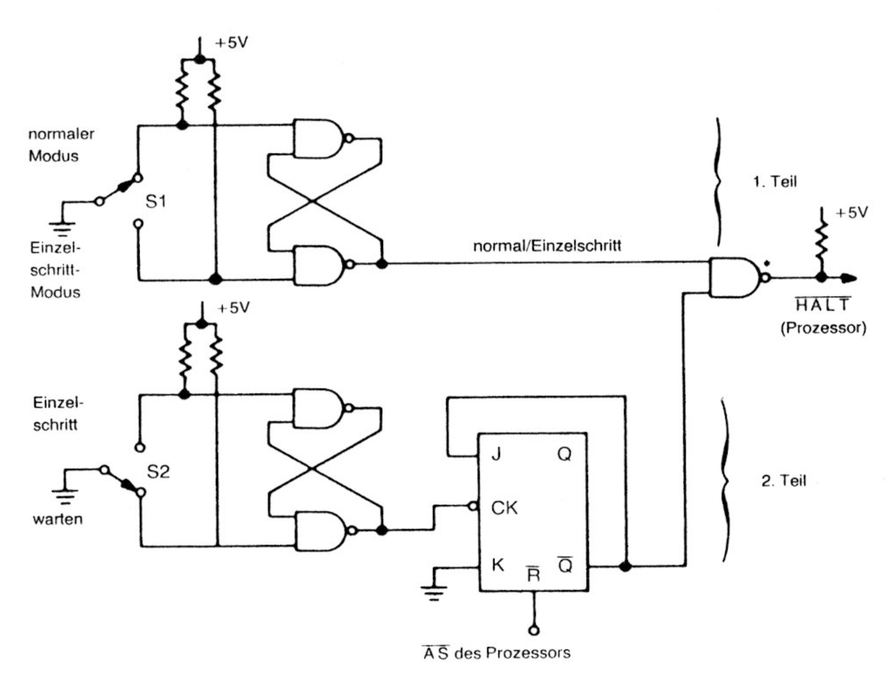


Abb. 2.18: Schaltung zur Realisierung des Einzelschritt-Betriebs

Das Zeitdiagramm (Abb. 2.19) macht diesen Ablauf deutlich.

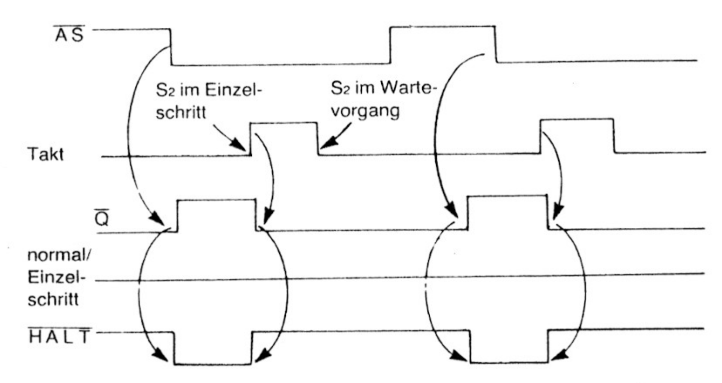


Abb. 2.19: Zeitdiagramm der Signale im Einzelschritt-Modus

Asynchrone Datenübertragung

Um im Asynchron-Modus die Steuerung des Buszyklus schrittweise zu betreiben, genügt es, das Signal $\overline{DTACK}$ um eine bestimmte Zeit T zu verzögern. Auf diese Weise wird der Prozessor Wartezustände Sw erzeugen, bis $\overline{DTACK}$ empfangen wird. Somit kann der Benutzer ungehindert den Zustand der Signale im Buszyklus untersuchen, bis er beispielsweise durch einen Knopfdruck das Signal $\overline{DTACK}$ sendet.

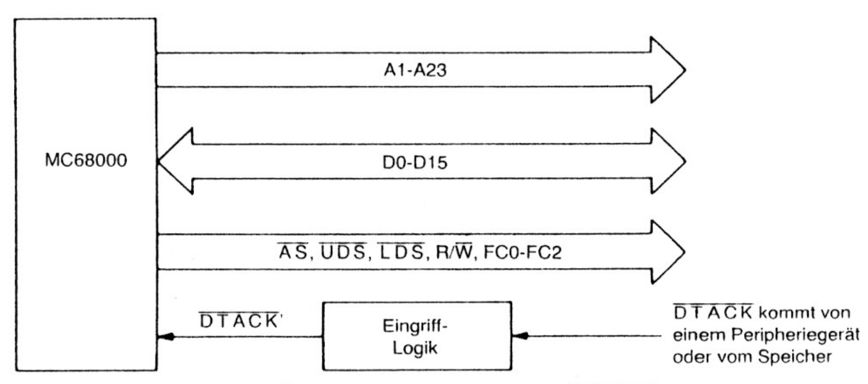
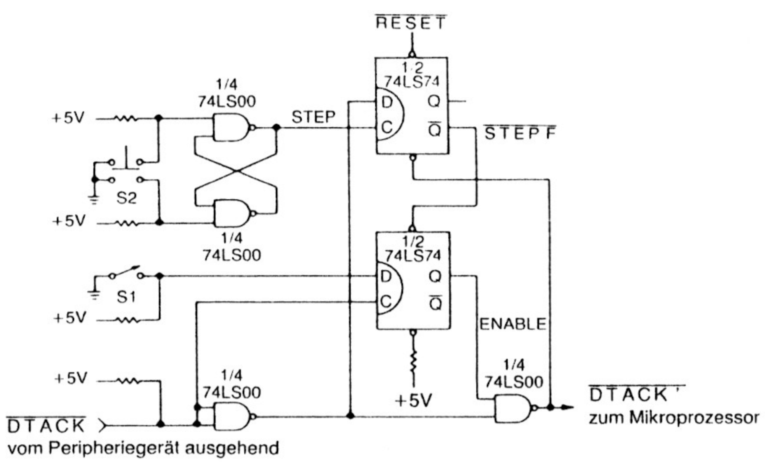


Abb. 2.20: Übersicht des Eingriffs in das $\overline{DTACK}$-Signal



S1 : offen : normale Durchführung
S1 : geschlossen : Einzelschritt-Modus
S2 : einmal hin- und hergeschaltet : bewirkt die Generierung eines Signals $\overline{DTACK}$

Abb. 2.21: Logik zur Realisierung des Einzelschritt-Modus durch Eingriff in das $\overline{DTACK}$-Signal

Es ist klar, daß in einer solchen Anwendung keine Zeitüberwachungsschaltung („Watchdog“) den Zyklus stören darf.

Die schematische Darstellung dieses Verfahrens ist in den Abb. 2.20 bis 2.22 zu finden.

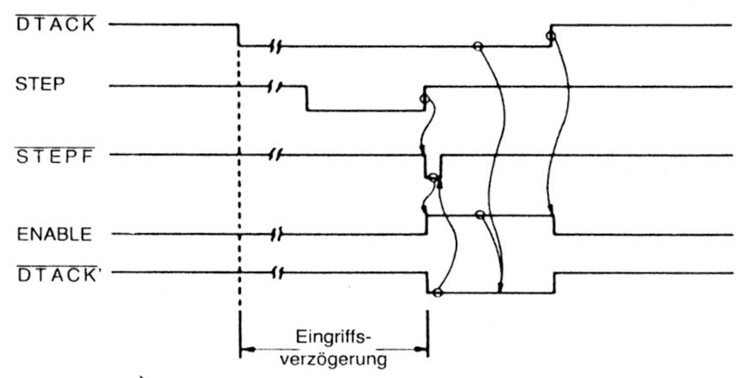


Abb. 2.22: Zeitdiagramm des Eingriffs in das DTACK-Signal

SYNCHRONE DATENÜBERTRAGUNG

Bei diesem Übertragungstyp gibt es keinen Dialog zwischen der übergeordneten Schaltung („Master“) und der untergeordneten („Slave“). Jedes Signal muß zwangsläufig innerhalb einer bestimmten Zeitspanne, die vom Prozessor festgelegt wird, eintreffen. Dieser Ablauf wird in dem Schema der Abb. 2.23 veranschaulicht.

Die gebräuchlichsten Peripheriebausteine aus der Familie des 6800 sind:

- 6821 Peripheral Interface Adapter (PIA)
  Universeller Parallel-Interface-Baustein
- 6840 Programmable Timer (PTM)
  Zeitgeberbaustein
- 6843 Steuerbaustein für Diskettenkontroller
- 6845 CRT-Controller (CRTC)
  Steuerbaustein für Bildschirme
- 6850 Asynchronous Communications Interface Adapter (ACIA)
  Asynchroner Datenübertragungsbaustein (UART)
- 6852 Synchronous Serial Data Adapter (SSDA)
  Synchroner Datenübertragungsbaustein

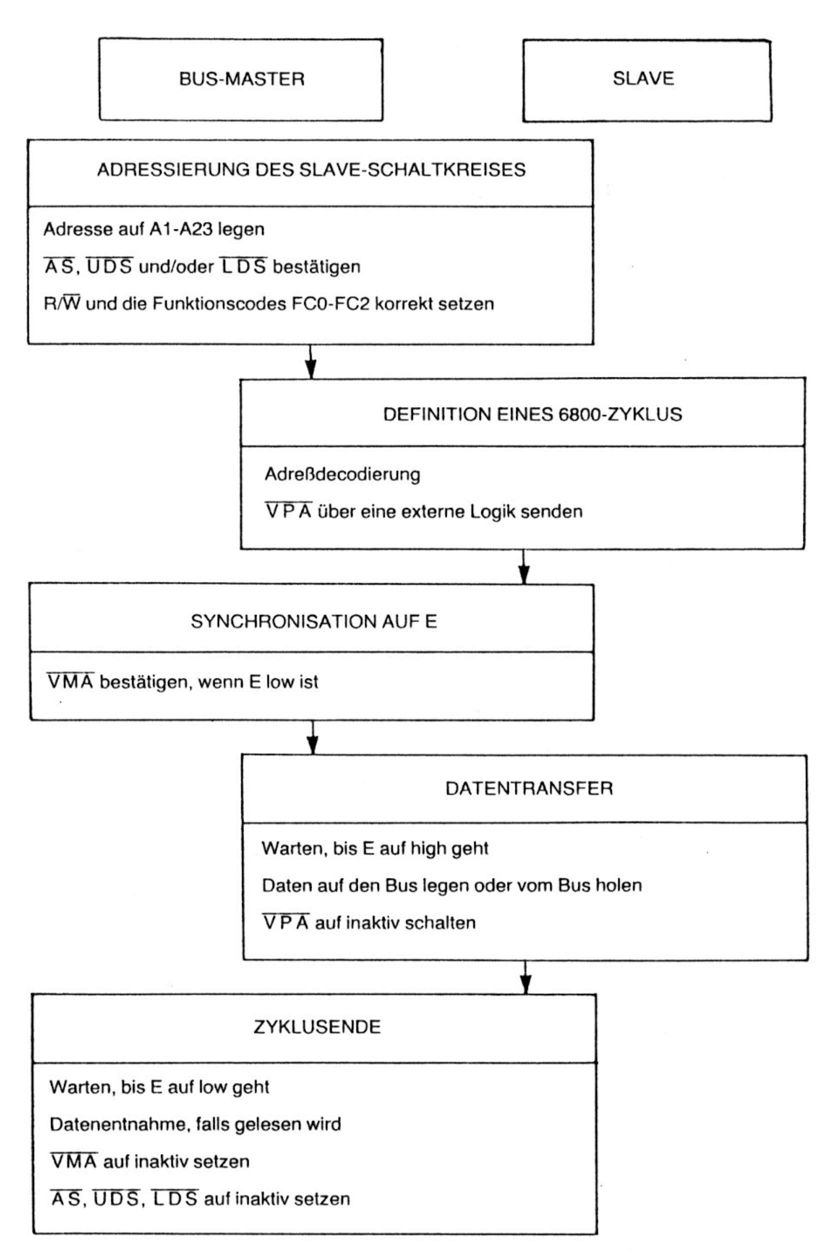


Abb. 2.23: Ablaufschema einer synchronen Datenübertragung

Um Daten zwischen dem MC68000 und einem Peripheriebaustein der 6800-Familie übertragen zu können, sind drei Steuersignale vorhanden:

E *(Enable) – Freigabe*
$\overline{\text{VPA}}$ *(Valid Peripheral Address) – gültige Peripherieadresse*
$\overline{\text{VMA}}$ *(Valid Memory Address) – gültige Speicheradresse*

Das Freigabesignal E entspricht dem E- oder Phi2-Signal des 6800-Systems. Dieser Takt wird von den 6800-Peripheriebausteinen verwendet, um den Datentransfer zu synchronisieren.

E ist ein Takt, der mit einer Frequenz von einem Zehntel des 68000-Systemtakts läuft. Die Taktperiode von E entspricht also 10 Taktperioden des 68000 mit – im Low-Zustand – 6 Taktperioden des Signals CLK und – im High-Zustand – 4 Taktperioden des Signals CLK (Abb. 2.24).

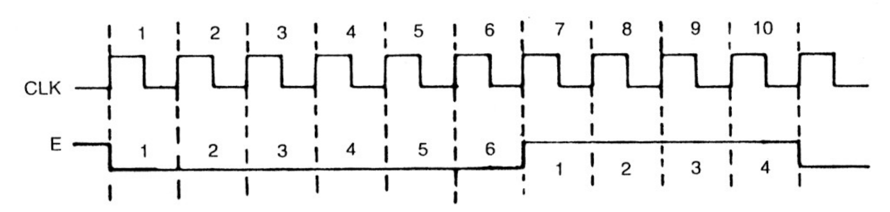


Abb. 2.24: Der E-Takt

Beispiel:

Im Falle des MC68000L8 hat der Takt eine Frequenz von 8 MHz und E etwa 0,8 MHz. Dies erlaubt also, die Peripheriebausteine der 1-MHz-Skala mit einem MC68000L8 zu verbinden.

Wir wollen uns nun das Zeitdiagramm für die synchrone Datenübertragung genauer anschauen.

Erläuterung des Zeitdiagramms (Abb. 2.25)

Erster Buszyklus: Lesezyklus im asynchronen Modus
Zweiter Buszyklus: Lesezyklus mit einem peripheren 6800-Baustein

Zustand S0:

– Der Adreßbus ist in hochohmigem Zustand.
– Die Funktionscodes sind gesetzt.

Zustand S1:

– Eine Adresse wird auf den Adreßbus gelegt.

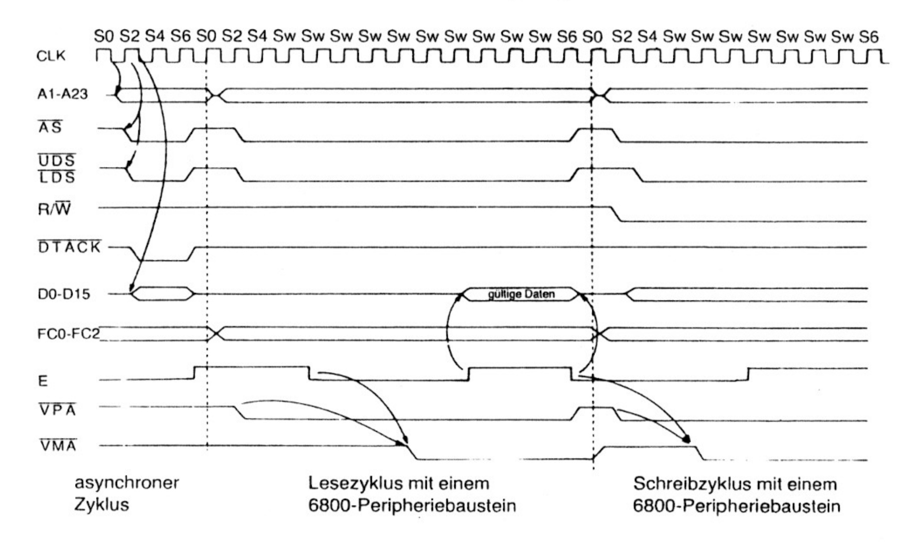


Abb. 2.25: Zeitdiagramm eines synchronen Zyklus

Zustand S2:

– $\overline{AS}$ wird aktiv, um anzugeben, daß sich eine gültige Adresse auf dem Bus befindet.
– $\overline{UDS}$ und $\overline{LDS}$ sind ebenfalls aktiv.

Zustände S3...Sw...S5:

– Der Prozessor, der das $\overline{VPA}$-Signal empfangen hat, weiß, daß ein synchroner Baustein adressiert wurde und daß die Operation durch das Freigabesignal E synchronisiert werden soll. Er wartet, bis E auf High gesetzt ist – evtl. indem er Wartezustände erzeugt –, um dann $\overline{VMA}$ zu aktivieren (low).

Das Signal $\overline{VMA}$ wird nun zur Selektion des peripheren Bausteins verwendet.

Während E auf High ist, stellt der Peripheriebaustein die zu lesende Information auf dem Datenbus zur Verfügung.

Zustand S6:

- Der Prozessor speichert die Daten.

Zustand S7:

- $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$ werden vom Prozessor zurückgesetzt (inaktiv, high).
- Das Signal E geht auf Low.
- Im nächsten Schritt wird der Adreßbus wieder in den hochohmigen Zustand versetzt.
- Die externe Logik setzt schließlich $\overline{VPA}$ zurück auf High (in einer Taktperiode), nachdem das Signal $\overline{AS}$ zurückgesetzt worden ist (auf High).

Dritter Buszyklus: Schreibzyklus mit einem peripheren 6800-Baustein

Zustand S0:

- Der Adreßbus ist in hochohmigem Zustand.
- Die Funktionscodes sind gesetzt.

Zustand S1:

- Eine Adresse wird auf den Adreßbus gelegt.

Zustand S2:

- $\overline{AS}$ wird bestätigt (low, aktiv), um anzugeben, daß sich eine gültige Adresse auf dem Bus befindet.
- R/$\overline{W}$ wird für den Schreibvorgang gesetzt (low).

Zustand S3:

- Die zu schreibenden Daten werden auf den Bus gelegt.

Zustand S4:

- $\overline{UDS}$, $\overline{LDS}$ werden bestätigt, um anzuzeigen, daß die sich auf dem Bus befindenden Daten gültig sind.

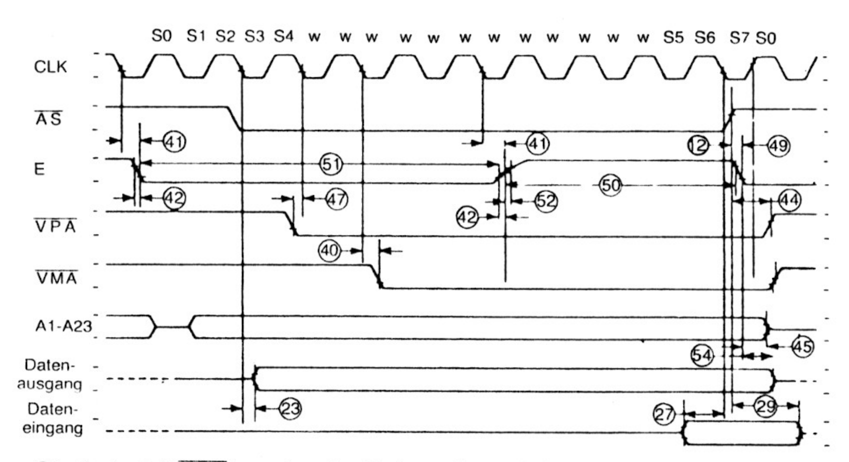
Zustände S5...Sw...S6:

- Der Prozessor hat das Signal $\overline{VPA}$ empfangen, kurz nachdem E auf Low gefallen ist und vor dem Eintreten des Zustands S4. Nach dem Empfang von $\overline{VPA}$ sendet der Prozessor $\overline{VMA}$ zum Peripheriebaustein. Die Daten werden während des High-Zustands von E angenommen.

Zustand S7:

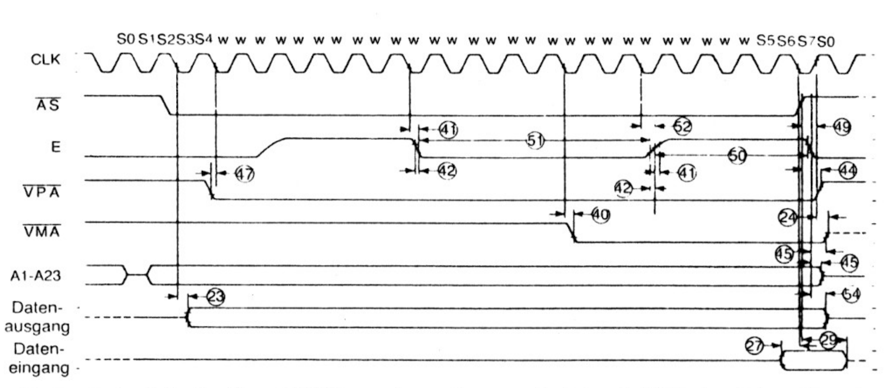
- R/$\overline{W}$ wird wieder auf High gesetzt.
- Der Datenbus wird hochohmig.
- Die externe Logik setzt $\overline{VPA}$ auf High zurück, sobald $\overline{AS}$ auf High zurückgesetzt worden ist.

Die Zeitdiagramme in Abb. 2.26 und 2.27 zeigen die Synchronisierungsprobleme im synchronen Modus.



Günstigster Fall: $\overline{VPA}$ kommt vor dem Ende von S4 an; die Setup-Zeit (47) wird berücksichtigt.

Abb. 2.26: Synchroner Zyklus



Ungünstigster Fall: Das Signal $\overline{VPA}$ kommt zu spät an, um die Setup-Zeit (47) einzuhalten. Es muß auf den nächsten E-Zyklus gewartet werden.

Abb. 2.27: Synchroner Zyklus

Die Kommunikation zwischen dem MC68000 und den Bausteinen der 6800-Familie ist dank der Signale E, $\overline{VPA}$ und $\overline{VMA}$ recht einfach. In Abb. 2.28 wird das Zusammenspiel verdeutlicht.

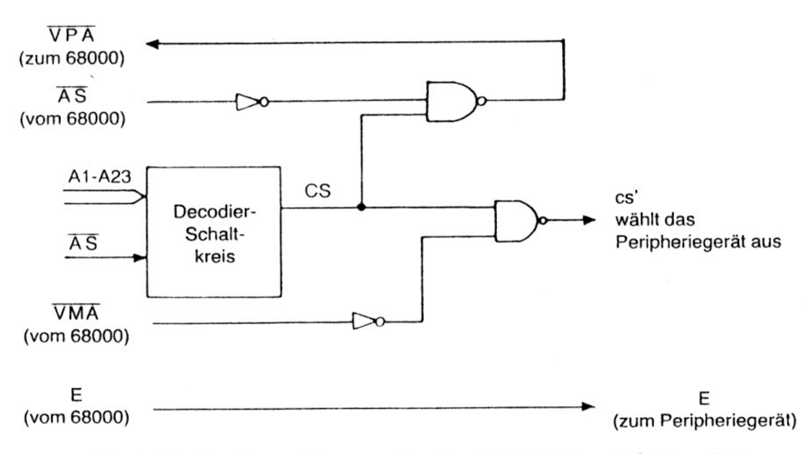


Abb. 2.28: Synchrone Kommunikation (MC68000 – 6800-Familie)

Wenn der Decodierbaustein ein Signal CS erzeugt und das Signal $\overline{AS}$ anliegt, wird automatisch ein Signal $\overline{VPA}$ erzeugt. Der 68000 wird nun davon unterrichtet, daß er mit einem im Synchron-Modus arbeitenden peripheren Baustein kommunizieren wird. Er sendet $\overline{VMA}$. Über die Verbindung von $\overline{VMA}$ und CS wird der periphere Baustein adressiert, der anschließend den Datentransfer vornehmen kann.

Der Schaltplan in Abb. 2.29 zeigt den synchronen Anschluß eines Bausteins der 6800-Familie, dem 6821, an den MC68000.

Einige Anwendungen benötigen aber einen anderen Takt als den E-Takt, z. B. wenn man den MC6854, einen Baustein für asynchrone Datenübertragungs-Steuerung, ADLC (Asynchronous Data Link Controller), ansprechen will, der Datenübertragungen mit hoher Geschwindigkeit durchführt. Diese Geschwindigkeit ist natürlich abhängig von dem verwendeten Taktgeber; in dem genannten Beispiel benötigt man einen Taktgeber mit einer Frequenz von mindestens 2 MHz.

Der synchrone Übertragungsmodus zwischen dem 6854 und dem 68000 würde den 6854 dazu zwingen, sich nach dem E-Takt zu richten. Dies wäre aber ein Widerspruch zur Funktion dieses Bausteins.

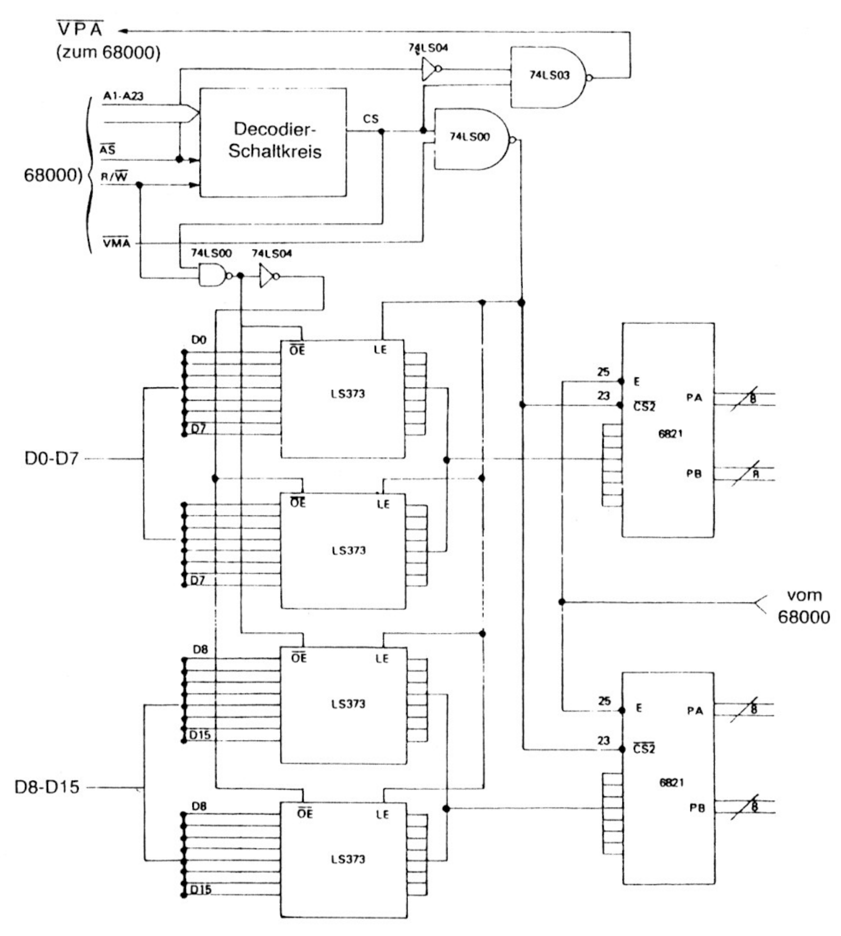


Abb. 2.29: Synchrone Verbindung zwischen dem MC68000 und 2 PIA 6821

Es muß also eine Möglichkeit geben, die 6800-Bausteine und den MC68000 im Asynchron-Modus zu verbinden.

Die Abb. 2.30 stellt eine schematische Übersicht eines solchen Verbindungstyps dar.

Puffer sind dazu da, um zu schreibende oder zu lesende Dateneinheiten zwischenzuspeichern. Ihre Arbeitsweise wird durch die Steuerlogik bestimmt, die von den Signalen R/$\overline{W}$ und CS abhängig ist.

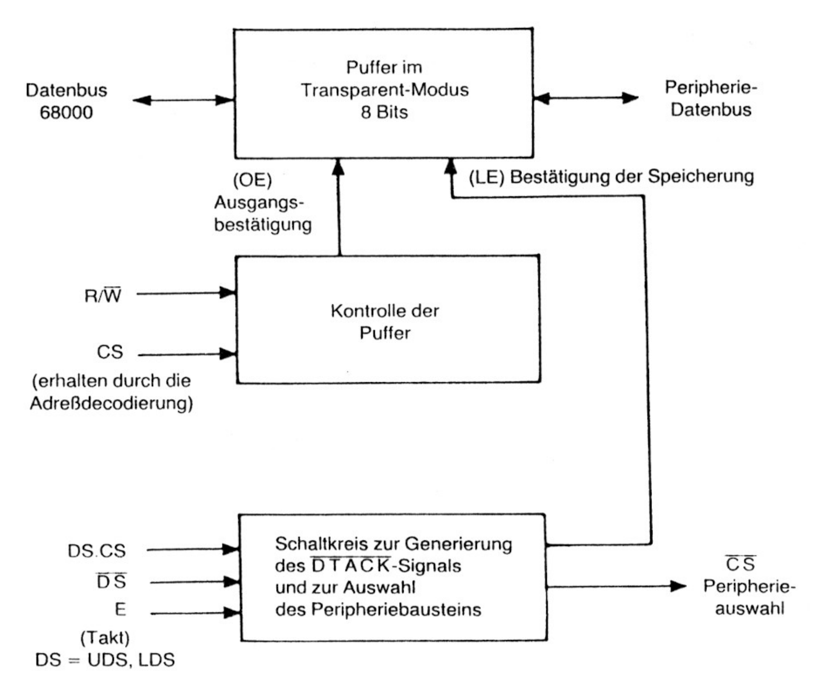


Abb. 2.30: Schematische Darstellung einer asynchronen Verbindungsschaltung

Eine Schaltung hat die Aufgabe, das Signal $\overline{DTACK}$ zu erzeugen, nachdem ein Peripheriebaustein ausgewählt wurde. Diese Schaltung ist von der Adreßbestätigung abhängig, die ihrerseits der Start für den Beginn des Buszyklus, der Decodierung und des Peripheriezeitgebers ist.

Um dieses Prinzip besser zu verstehen, betrachten wir das Schema in Abb. 2.31 im einzelnen.

Zuerst werden die Flip-Flops U1A und U1B auf 0 zurückgesetzt ($\overline{UDS}$ oder $\overline{LDS}$=H zieht $\overline{DS}$=H nach sich).

Dies setzt $\overline{DTACK}$ in den High-Zustand, was den Puffern einen Transparent-Modus auferlegt.

Der Puffer U2 ist hochohmig, da $\overline{OE}$=H ist. Der Puffer U3 ist durchgeschaltet, da $\overline{OE}$=L ist.

Wenn es sich um einen Schreibvorgang handelt, bleibt der Puffer U3 zu Beginn des Zugriffszyklus durchgeschaltet. Handelt es sich um einen Lesevorgang – R/$\overline{W}$=H, CS=H, und der Ausgang von U4A ist low –

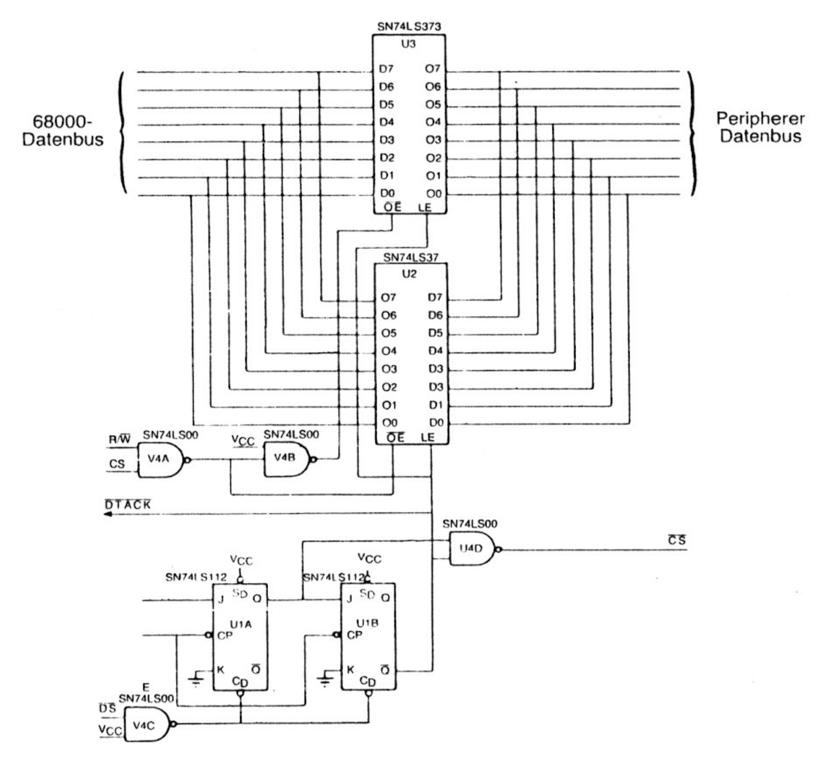


Abb. 2.31: Asynchrone Verbindung zwischen dem MC68000 und einem 6800-Peripheriebaustein

wird U2 durchgeschaltet, und U3 wird in den hochohmigen Zustand versetzt. Um den Zugriff zum peripheren Baustein zu ermöglichen, ist es notwendig, daß $\overline{UDS}$ oder $\overline{LDS}$ =L sind und folglich DS=H und CS=H.

Auf diese Weise wird sich der Ausgang Q des Flip-Flops U1A an der ersten fallenden Flanke des Takts E im High-Zustand befinden, um den Peripheriebaustein zu aktivieren. An der zweiten fallenden Flanke von E geht der Ausgang $\overline{Q}$ des Flip-Flops U1B in den Low-Zustand über. Das Signal $\overline{DTACK}$ wird gesendet, und die Daten werden in dem entsprechenden Puffer gespeichert. Das Signal $\overline{DTACK}$ wird in U4D invertiert, was das Peripheriegerät freigibt. Schließlich, wenn $\overline{UDS}$ oder $\overline{LDS}$ wieder auf High gesetzt werden und DS=H ist, werden die Flip-Flops U1A und U1B auf 0 zurückgesetzt und die Schaltungen auf den nächsten Zugriff vorbereitet.

Die Schaltpläne in Abb. 2.32 und 2.33 stellen Anwendungen dieses Prinzips vor.

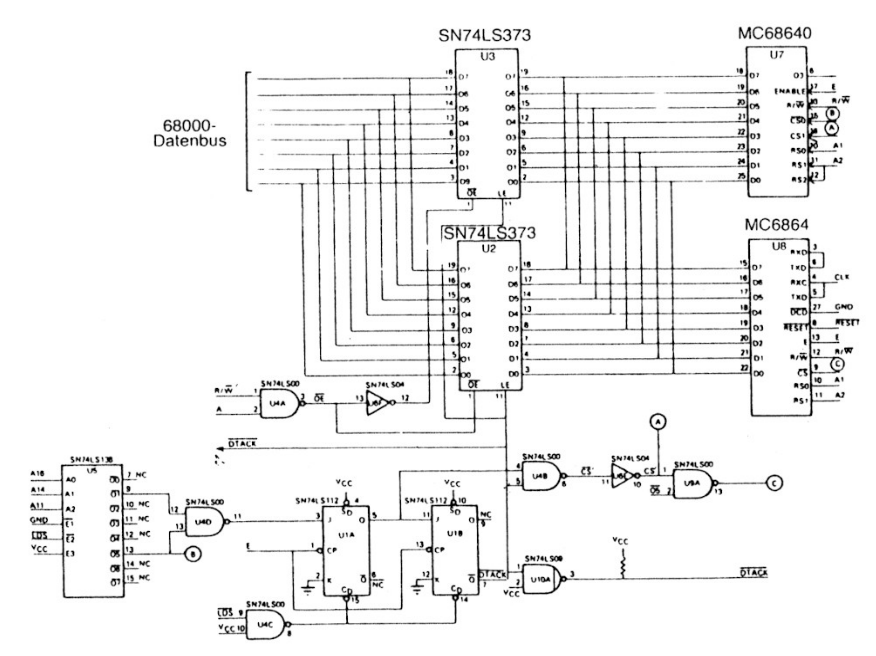


Abb. 2.32: Verbindung zwischen dem MC68000, einem programmierbaren Zeitgeber (6840) und einem Baustein für asynchrone Datenübertragungs-Steuerung (6854) im Asynchron-Modus

BUSZUWEISUNGSSTEUERUNG

Die Buszuweisungssteuerung ermöglicht einem externen Baustein, der fähig ist, die Bussteuerung zu übernehmen, eine Busanforderung zu stellen, danach eine Bestätigung zu erhalten und schließlich die Kontrolle selbst zu bekommen.

In der Praxis läuft das so ab:

- Der externe Baustein stellt eine Busanforderung – $\overline{BR}$ (Bus Request).

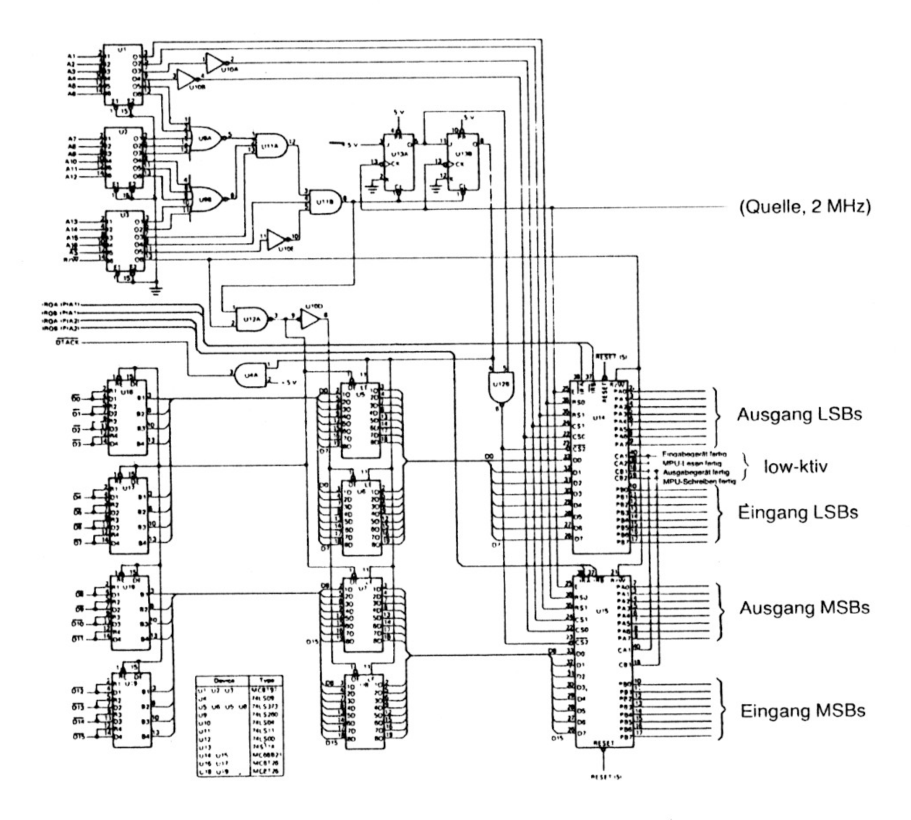


Abb. 2.33: Verbindung zwischen dem MC68000 und zwei PIA (6821)

- Empfang der Anforderung und Bestätigung, daß der Bus am Ende des ablaufenden Zyklus frei wird – $\overline{\text{BG}}$ (Bus Grant).
- Der externe Baustein gibt an den Prozessor das Signal für die Busfreigabeerkennung zurück – $\overline{\text{BGACK}}$ (Bus Grant ACKnowledge).

Dieser Steuerungstyp ermöglicht die Realisierung von Multiprozessor-Anwendungen: CPU, DMAC, MMU etc., bei denen jeder mögliche Verwalter die Kontrolle übernehmen kann.

Der Ablauf bei der Übergabe der Buskontrolle ist in Abb. 2.34 wiedergegeben.

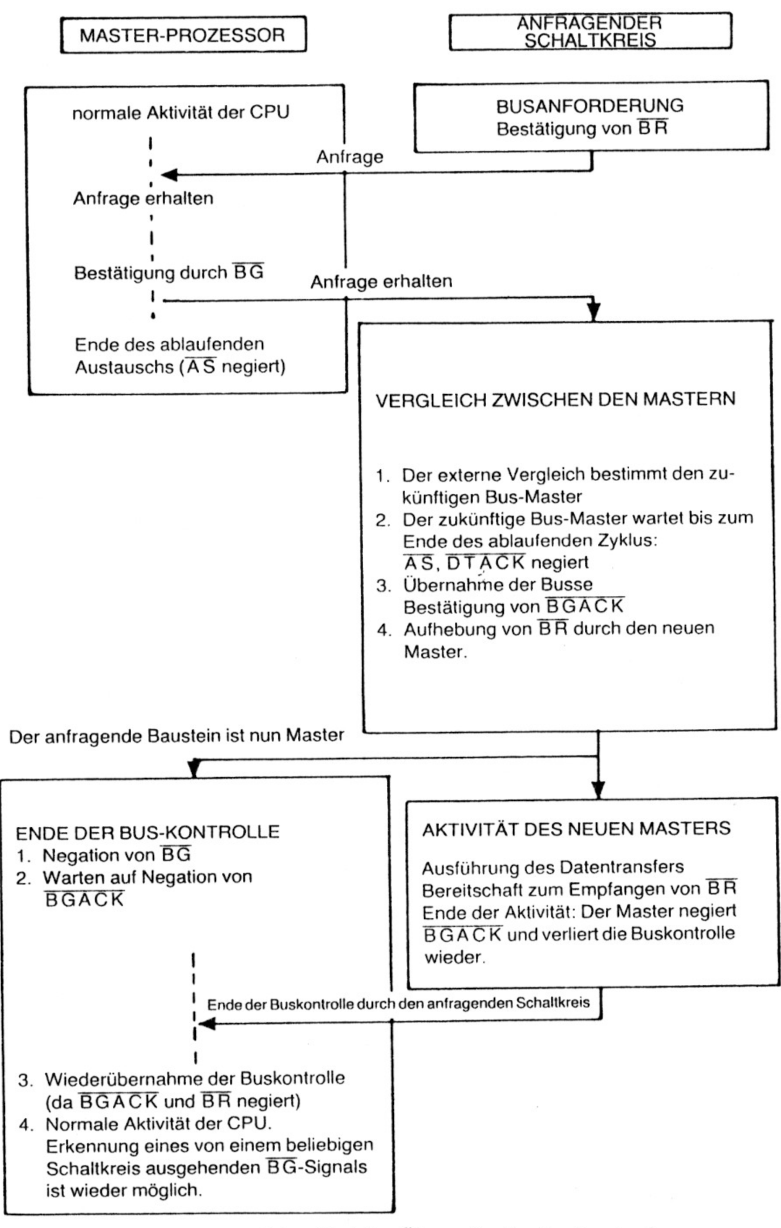


Abb. 2.34: Ablauf bei der Übergabe der Buskontrolle

Erläuterung des Ablaufschemas (Abb. 2.34)

Busanforderung

Ein externer Baustein, der in der Lage ist, die Verwaltung der Busse zu übernehmen, fordert die Buskontrolle durch Senden des Signals $\overline{BR}$. Dadurch wird der Prozessor darauf hingewiesen, daß ein externer Baustein die Bussteuerung zu übernehmen wünscht. Der 68000 hat immer eine geringere Buspriorität als der anfragende Baustein, und er übergibt die Buskontrolle, sobald er den letzten angefangenen Buszyklus beendet hat.

Wenn er vor dem Zurücksetzen des Signals $\overline{BR}$ keine Busfreigabeerkennung ($\overline{BGACK}$) empfängt, fährt der Prozessor unbeirrt mit seiner Arbeit fort, bis $\overline{BR}$ zurückgesetzt wird. Dies ermöglicht die Fortsetzung der Verarbeitung in dem Fall, wo der anfragende Baustein vielleicht aus Unachtsamkeit einer anhängenden Schaltung geantwortet hat.

Einverständnis des Prozessors

Der Prozessor erkennt die Busanforderung so bald wie möglich an, indem er mit $\overline{BG}$ bestätigt. Normalerweise findet das sofort nach der internen Synchronisierung von $\overline{BG}$ statt. Die einzige Ausnahme dieser Regel ist, wenn der Prozessor, nachdem die interne Entscheidung, den nächsten

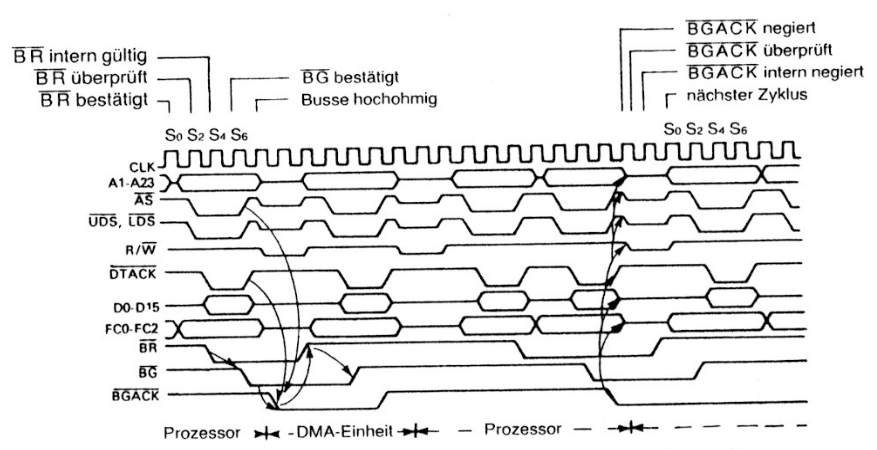


Abb. 2.35: Zeitdiagramm für die Steuerung der Buskontrolle

Zyklus durchzuführen, getroffen ist, nicht weit genug im Zyklus fortgeschritten ist, um $\overline{AS}$ zu senden. Die Bestätigung $\overline{BG}$ findet also immer einen Taktzyklus nach dem Senden von $\overline{AS}$ statt.

Das Zeitdiagramm in Abb. 2.36 zeigt einen solchen Fall.

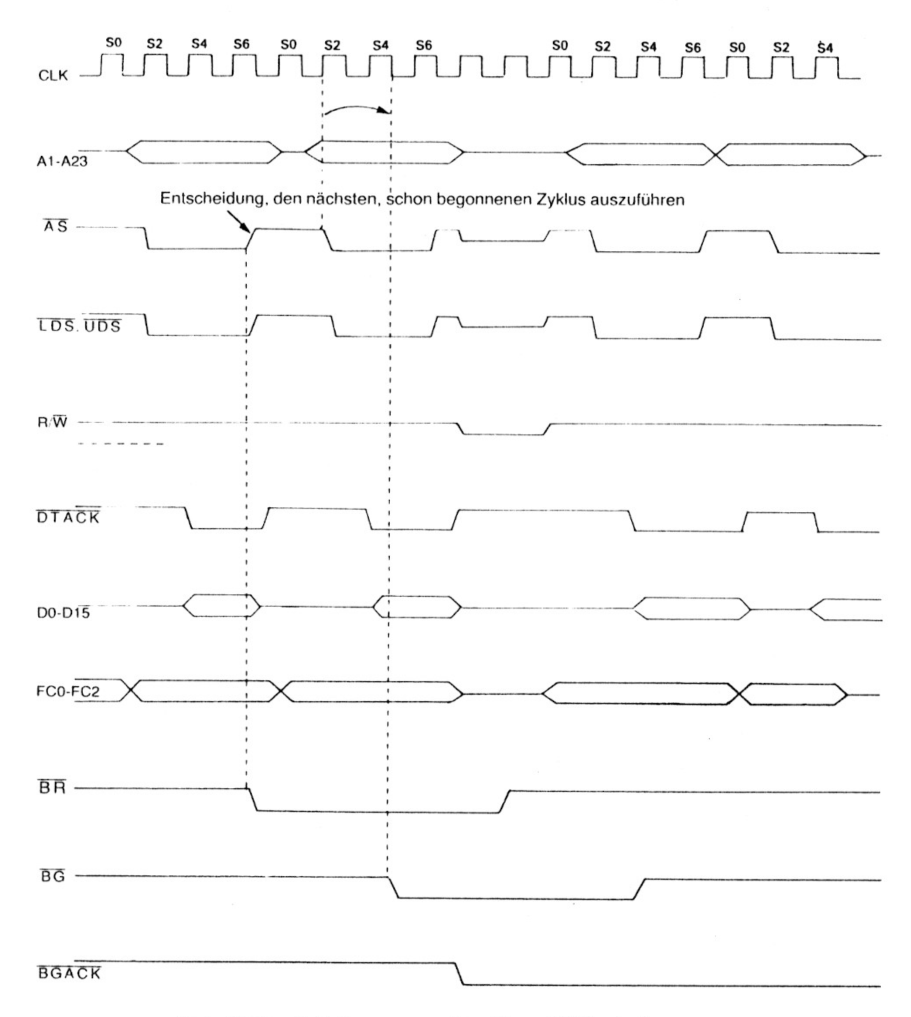


Abb. 2.36: Zeitdiagramm: Das Signal $\overline{BG}$ wird verzögert

Busfreigabeerkennung

Wenn der anfragende Baustein $\overline{BG}$ empfängt, wartet er zunächst darauf, daß

$\overline{AS}$
$\overline{DTACK}$
$\overline{BGACK}$

zurückgesetzt worden sind, bevor er das Signal $\overline{BGACK}$ sendet.

- Das Zurücksetzen von $\overline{AS}$ deutet darauf hin, daß der vorhergegangene Bus-Master seinen Zyklus beendet hat.
- Das Zurücksetzen von $\overline{DTACK}$ bedeutet, daß die entsprechende untergeordnete Schaltung ihre Aufgabe beendet und die Verbindung unterbrochen hat. In einigen Anwendungen darf allerdings das Signal $\overline{DTACK}$ überhaupt nicht benutzt werden. Die universellen Schaltungen müssen also so verkoppelt sein, daß sie nur von $\overline{AS}$ abhängig sind.
- Das Zurücksetzen von $\overline{BGACK}$ gibt dem anfragenden Baustein an, daß der vorhergehende Bus-Master die Busse freigegeben hat. Wenn er das Signal $\overline{BGACK}$ gesendet hat, ist der Baustein Master, bis er selbst $\overline{BGACK}$ zurücksetzt, nachdem er seinen Zyklus vollständig beendet hat.

Beispiel der Buszuweisungssteuerung zwischen dem 68000 und dem DMAC 6844

Der DMA-Controller DMAC (Direct Memory Access Controller) besitzt vier Übertragungskanäle. Mit jedem Kanal sind verbunden:

- ein Eingang TxRQ, der von einem peripheren Baustein gesetzt wird, der eine DMA-Verarbeitung wünscht;
- ein 16-Bit-Adreßregister, in dem die Startadresse des zu übertragenden Datenblocks gespeichert wird;
- ein 16-Bit-Zähler, der die Anzahl der zu übertragenden Bytes aufnimmt.

Dieser Baustein besitzt verschiedene Betriebsarten. Es genügt jedoch zu wissen, daß, wenn eine Übertragungsanforderung vorliegt (TxRQ=H), der DMAC das Ausgangssignal $\overline{DRQH}$ sendet. Danach wartet er auf die Anerkennung der Anfrage, in dem Fall auf den Empfang des Signals DGRNT. In diesem Moment beginnt die Datenübertragung. Am Ende dieser Übertragung wird der Ausgang $\overline{TxSTB}$ gesetzt und der Ausgang $\overline{DRQH}$ zurückgesetzt.

Der Schaltplan dieser Verbindung ist in Abb. 2.37 zu sehen.

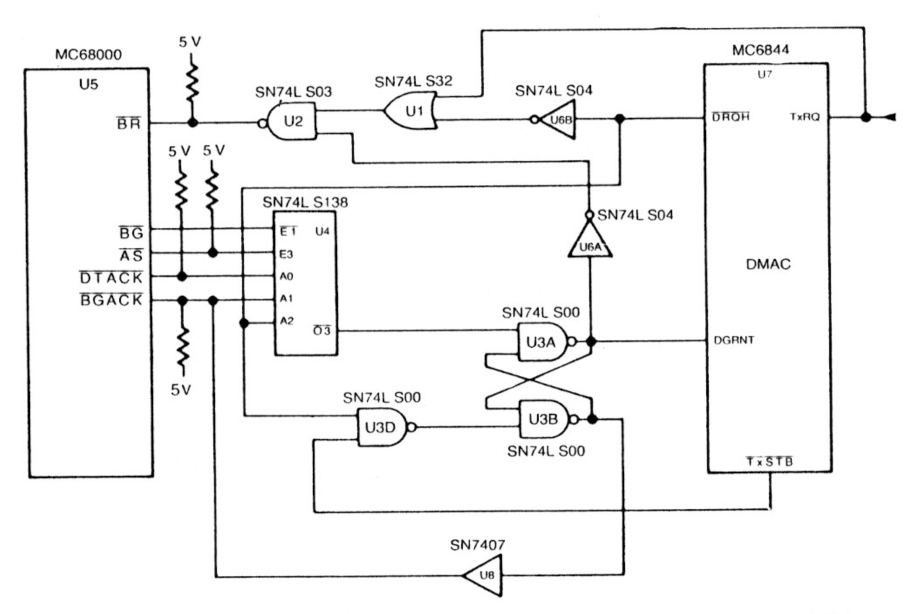


Abb. 2.37: Schaltung für die Verbindung eines DMAC 6844 mit dem MC68000

Erläuterung der Schaltung (Abb. 2.37)

Das System ist im Ruhezustand, also $\overline{DRQH}$=1, DGRNT=0 und $\overline{TxSTB}$=1. Der Eingang von U2, der von U6A kommt, ist demzufolge 1. Ein DMA-Gesuch erreicht den Controller: Über U1 wird der zweite Eingang von U2 auf 1 gesetzt, der Ausgang sendet 0. Das Signal $\overline{BR}$ wird gesendet. Der 68000 bestätigt mit $\overline{BG}$ und gibt die Busse frei. Der DMAC antwortet, indem er durch Senden von $\overline{DRQH}$ bestätigt. Die anderen Signale haben folgende Werte:

$\overline{BGACK}$=1
$\overline{DTACK}$=1
$\overline{AS}$=1
$\overline{BG}$=0

Der Decodierbaustein U4 kann seinen Ausgang $\overline{O3}$ setzen, was bewirkt, daß DGRNT gesendet wird (DGRNT=1) und $\overline{BR}$ über U6A und U2 zurückgesetzt wird.

Der Baustein DMAC kann seine Datenübertragungen beginnen. Außerdem muß aber noch $\overline{BGACK}$ gesetzt werden. $\overline{DRQH}$ ist 0 und $\overline{TxSTB}$ ist 1, also ist der Ausgang U3D=1. Folglich ist DGRNT=1 und U3B=0.

$\overline{BGACK}$ kann somit gesendet werden. Der 68000 setzt kurz danach $\overline{BG}$ zurück.

Das Ende der Kontrolle durch den DMAC-Baustein läuft folgendermaßen ab: Wenn der Datentransfer beendet ist, wird der Ausgang $\overline{TxSTB}$ auf 0 gesetzt. Die Flip-Flops U3A und U3B werden also wieder auf 0 zurückgesetzt, da $\overline{DRQH}$ vom DMAC intern zurückgesetzt worden ist. Ist $\overline{DRQH}=1$, geht der Ausgang U3D auf 0, der Ausgang U3B auf 1, und $\overline{BGACK}$ ist zurückgesetzt.

Ebenso ist $\overline{O3}=1$ und $\overline{BGACK}=1$, dann ist DGRNT=0.

Übungen

2.1: Wie lange dauert ein Lese- bzw. Schreibzyklus mindestens?

2.2: In welcher Reihenfolge werden die folgenden Signale oder Busse während eines Lese- und eines Schreibzyklus gesetzt?

Adreßbus
$\overline{AS}$
$\overline{UDS}$
$\overline{LDS}$
$R/\overline{W}$

2.3: Welche besondere Eigenschaft zeichnet einen Lese-/Änderungs-/Schreibzyklus aus?

2.4: Welche Voraussetzungen sind für den Neustart eines Zyklus nötig?

2.5: Welche drei Voraussetzungen sind für die Bestätigung des Signals $\overline{BGACK}$ nötig. Erklären Sie sie.

Lösungen

2.1: Die minimale Dauer eines Lese- bzw. Schreibzyklus beträgt vier Takte.

2.2: Bei einem Lesevorgang ist es folgendermaßen:

- Setzen von $R/\overline{W}$.
- Ausgabe der Adresse auf A1–A23.
- Bestätigung der Adreßgültigkeit $\overline{AS}$.
- Setzen von $\overline{UDS}$ oder/und $\overline{LDS}$.

Bei einem Schreibvorgang läuft es so ab:

- Ausgabe der Adresse auf A1–A23.
- Bestätigung durch $\overline{AS}$.
- Setzen von R/$\overline{W}$.
- Setzen von $\overline{UDS}$, $\overline{LDS}$ (nachdem die zu schreibenden Daten auf den Datenbus gelegt worden sind).

2.3: Während dieses Vorgangs bleibt das Signal $\overline{AS}$ die ganze Zeit gesetzt. Das verhindert den Eingriff einer externen Schaltung für die Dauer eines Zyklus.

2.4: Um den Neustart eines Zyklus zu bewirken, müssen die Signale $\overline{BERR}$ und $\overline{HALT}$ gesendet werden.

2.5: Damit ein anfragender Baustein die Busfreigabeerkennung senden kann, müssen die Signale $\overline{AS}$, $\overline{DTACK}$ und $\overline{BGACK}$ zurückgesetzt worden sein. D. h.:

- Der Prozessor benötigt die Busse nicht mehr.
- Kein anderer peripherer Baustein hat eine Anforderung gestellt.
- Kein anderer Prozessor hat die Buskontrolle.

Kapitel 3

Die Ausnahmezustände

ALLGEMEINE BETRACHTUNGEN

Der MC68000 befindet sich immer in einem der folgenden Zustände:

- Normalzustand
- Ausnahmezustand
- Haltezustand

– Der Normalzustand entspricht der Ausführung von „normalen“ Befehlen, d. h. Befehlen, die keine Unterbrechungs- oder Ausnahmeprozeduren erzeugen.

Ein Spezialfall des Normalzustandes ist derjenige, der durch den Befehl STOP erzeugt wird: Der Prozessor befindet sich im „gestoppten“ Zustand.

– Der Ausnahmezustand ist die Antwort des Prozessors auf Unterbrechungen, Ausführung des Befehls TRAP und anderer Abweichungen im Programmablauf.

Die Ausnahme wird intern durch einen Befehl oder eine außergewöhnliche Bedingung, die während der Ausführung eines Befehls eintritt, erzeugt. Extern wird sie durch einen Befehl, einen Busfehler oder RESET generiert.

– Der Haltezustand weist auf einen schwerwiegenden Hardware-Fehler hin.

Tritt beispielsweise während der Ausführung einer Busfehler-Ausnahmeprozedur ein zweiter Busfehler auf, so kommt der Prozessor zum Stillstand. Nur ein externes RESET kann den Wiederanlauf bewirken.

Achtung: Der Haltezustand ist nicht mit dem STOP-Zustand zu verwechseln.

Der Prozessor kann zwei verschiedene Betriebsmodi annehmen:

- den Anwendermodus und
- den Überwachungsmodus, auch privilegierter oder Supervisor-Modus genannt.

Die Existenz eines privilegierten Modus gibt dem System eine große Sicherheit: Die Anwenderprogramme können nicht auf Informationen zugreifen, die nicht verändert werden dürfen.

Der Überwachungsmodus

- Der Zustand wird durch das Bit S des Statusregisters SR bestimmt (S=1).
- Alle Befehle stehen zur Verfügung.
- Der verwendete Stapelzeiger ist der Zeiger des Überwachungsstapels SSP (Supervisor Stack Pointer).

Der Anwendermodus

- Der Zustand wird ebenfalls durch das Bit S des Statusregisters SR bestimmt (S=0).
- Die Befehlsskala ist beschränkt: Verboten sind beispielsweise die Befehle STOP und RESET sowie die Befehle, die das Statusregister und die Transfers zum oder vom Anwenderstapelzeiger verändern.

Wechsel zwischen Anwender- und Überwachungsmodus

Wenn sich der Prozessor im Anwendermodus befindet, können nur Ausnahmeprozeduren einen Übergang in den Überwachungsmodus bewirken (Abb. 3.1).

Der Übergang vom Überwachungs- in den Anwendermodus kann durch vier verschiedene Befehle erreicht werden:

- RTE (Rückkehr aus einem Ausnahmezustand)
- MOVE Wort ins SR (Transfer eines Wortes in das Statusregister)

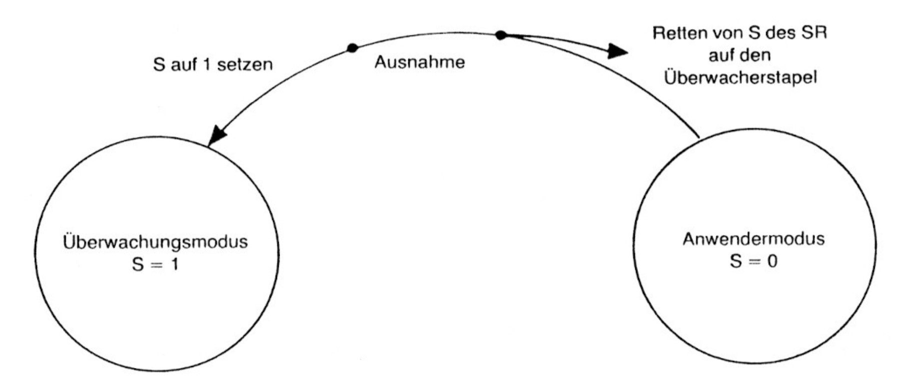


Abb. 3.1: Übergang vom Anwender- in den Überwachungsmodus

- ANDI mit SR (logisches UND unmittelbar)
- EORI mit SR (Exklusives ODER unmittelbar)

Der nächste Befehl wird folglich in dem Zustand ausgeführt, der durch den neuen Wert von S festgelegt worden ist (Abb. 3.2).

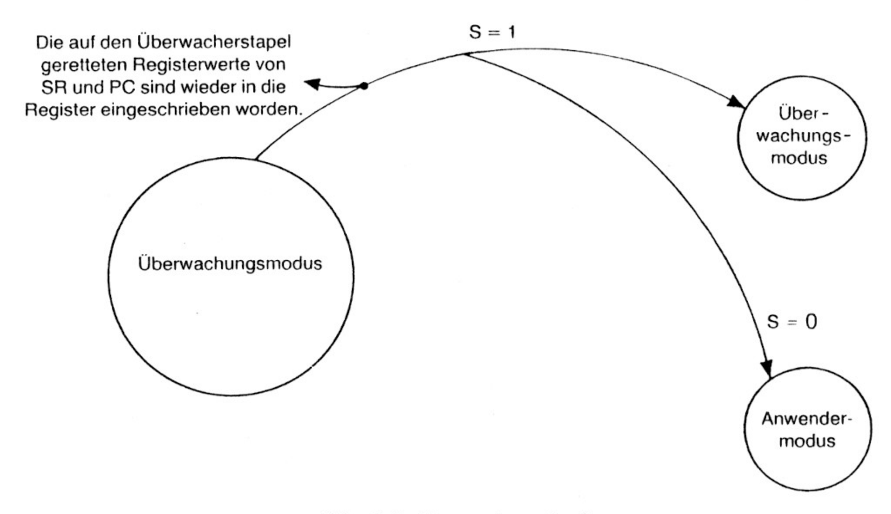


Abb. 3.2: Zustandswechsel

DIE AUSNAHMEVEKTOREN

Ausnahmevektoren sind Informationen, die sich im Speicher befinden und die dem Prozessor die Möglichkeit geben, die Ausnahmeprozeduren in Gang zu bringen. Alle Ausnahmevektoren sind zwei Worte lang (32 Bit), ausgenommen der Vektor RESET, der sich auf vier Worte erstreckt.

Die Vektoren befinden sich im Speicherbereich „Supervisor-Daten“, außer dem RESET-Vektor, der zu den „Supervisor-Programmen“ zählt.

Die Vektornummer gibt die Adresse eines Vektors an. Diese Nummer wird intern oder extern gemäß der Ausnahme erzeugt. Beispielsweise wird die Vektornummer der Ausnahme $\overline{BERR}$ intern berechnet, und zwar ist sie 2. Die Nummer einer Unterbrechung, die vom Baustein DMAC erzeugt wird, ist hingegen von dem externen Gerät geliefert worden. Bei Unterbrechungen liefert der periphere Baustein während der Anerkennungsphase der Unterbrechung dem Prozessor eine 8-Bit-Vektornummer über die Leitungen D0–D7.

Der Prozessor wandelt die 8-Bit-Zahl in eine 24-Bit-Adresse um, indem er sie mit 4 multipliziert.

Aufbau der Vektornummer:

D15								D7							D0
X	X	X	X	X	X	X	X	V7	V6	V5	V4	V3	V2	V1	V0

Aufbau der umgerechneten Adresse:

A23			A9									A0
0	0	0........	V7	V6	V5	V4	V3	V2	V1	V0	0	0

Die Vektoren werden in einer Tabelle verwaltet, wie in Abb. 3.3 zu sehen ist.

Nummer des Vektors	Adresse Dez.	Hex.	Betroffene Ausnahme
0	0	000	Reset: Initialisierung von SSP
	4	004	Reset: Initialisierung von PC
2	8	008	Busfehler
3	12	00C	Adreßfehler
4	16	010	Illegaler Befehl
5	20	014	Division durch Null
6	24	018	Befehl CHK
7	28	01C	Befehl TRAPV
8	32	020	Privilegverletzung
9	36	024	Ablaufverfolgung
10	40	028	Befehl mit 1010 am Anfang
11	44	02C	Befehl mit 1111 am Anfang
12	48	030	Reserviert
13	52	034	Reserviert
14	56	038	Reserviert
15	60	03C	Nicht initialisierter Unterbrechungsvektor
16-23	64	040	Reserviert
	95	05F	
24	96	060	Unechte Unterbrechung
25	100	064	autovektorielle Unterbrechung, Ebene 1
26	104	068	autovektorielle Unterbrechung, Ebene 2
27	108	06C	autovektorielle Unterbrechung, Ebene 3
28	112	070	autovektorielle Unterbrechung, Ebene 4
29	116	074	autovektorielle Unterbrechung, Ebene 5
30	120	078	autovektorielle Unterbrechung, Ebene 6
31	124	07C	autovektorielle Unterbrechung, Ebene 7
32–47	128	080	TRAP-Befehlsvektoren
	191	0BF	–
48–63	192	0C0	Reserviert
	255	0FF	–
64–255	256	100	Anwender-Unterbrechungsvektoren
	1023	3FF	–

Abb. 3.3: Tabelle der Ausnahmevektoren

Nachdem wir nun das Prinzip der Ausnahmevektoren erklärt haben, wollen wir im folgenden Abschnitt anhand eines Ablaufschemas die Abarbeitung einer Ausnahme kommentieren (Abb. 3.4).

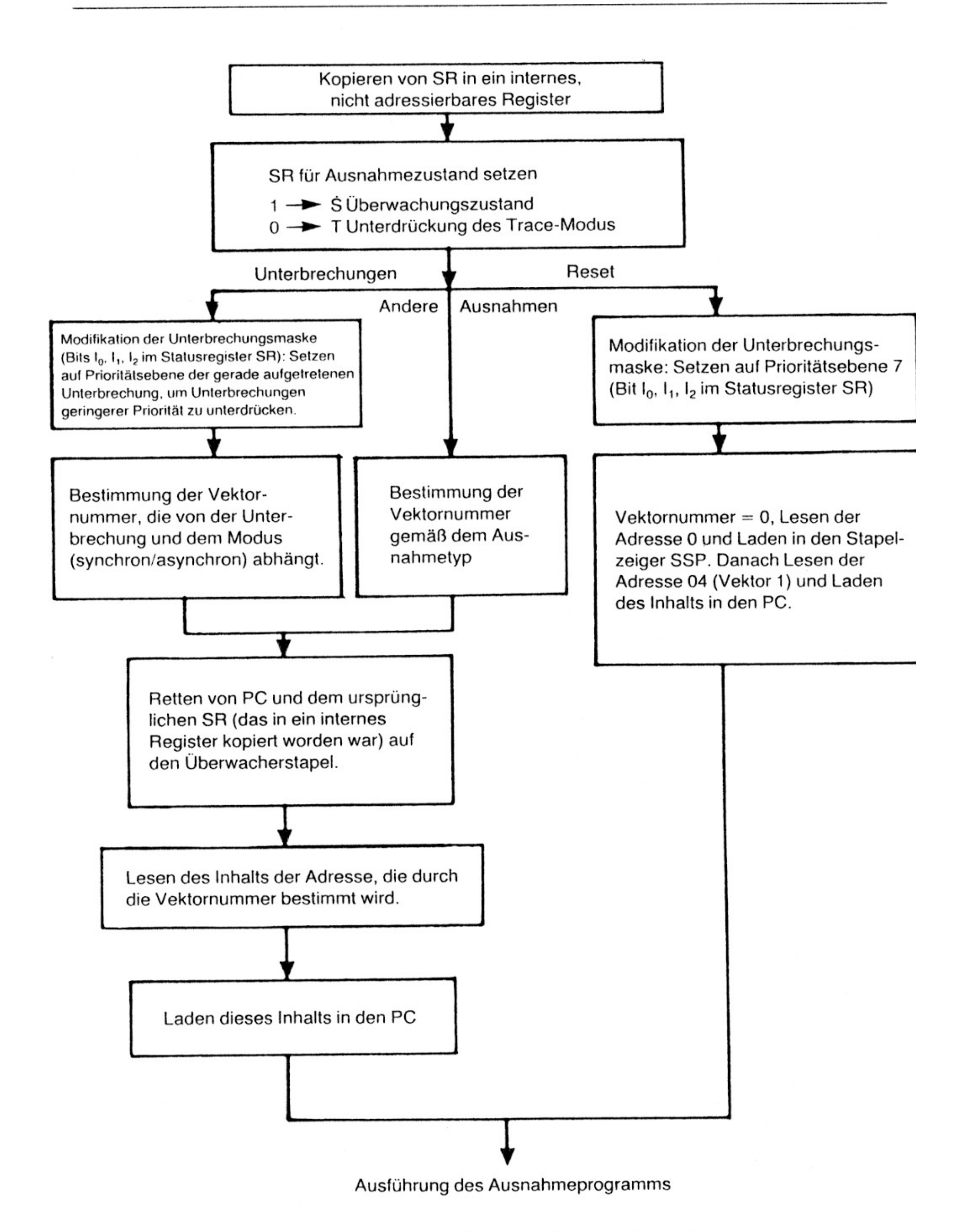


Abb. 3.4: Ablaufschema der Verarbeitung einer Ausnahme

VERARBEITUNG EINER AUSNAHME

Erster Schritt

Der Inhalt des Statusregisters wird als Kopie in einem internen Register abgelegt. Dann wird das Bit S gesetzt, also der Prozessor in den Überwachungsstatus versetzt. Das Bit T wird auf 0 gesetzt. Dies ermöglicht der Ausnahmeprozedur, ohne Behinderung durch den Trace-Modus (Trace ist eine Ausnahme) abzulaufen.

Entsprechend der jeweiligen Unterbrechung und des Reset, wird die Unterbrechungsmaske mit der entsprechenden Prioritätsebene versehen (Ebene 7 für RESET).

Zweiter Schritt

Die Vektornummer der Ausnahme wird bestimmt.

- Bei Unterbrechungen wird dem Prozessor die Nummer während der Anerkennungsphase der Unterbrechung geliefert.
- Bei den anderen Ausnahmen, den autovektoriellen Unterbrechungen, bestimmt die interne Logik des Prozessors die Vektornummer.

Dritter Schritt

Der derzeitige Zustand des Prozessors wird außer beim Reset gerettet.

Der aktuelle Wert des Programmzählers PC und die gesicherte Kopie des Statusregisters SR werden in den Überwachungsstapel geschrieben. Der Inhalt des Programmzählers ist im allgemeinen die Adresse des nächsten Befehls, der ausgeführt worden wäre, wenn keine Ausnahme eingetreten wäre. Dennoch ist der in den Stapel gesetzte Wert des PC im Falle eines Bus- oder Adreßfehlers unvorhersehbar (aufgrund des Vorgriffphänomens) und kann um einen bestimmten Betrag höher sein als die Adresse des Befehls, der den Fehler verursacht hat. Auf jeden Fall werden bei Bus- und Adreßfehlern zusätzliche Informationen, die den aktuellen Zustand dokumentieren, in den Stapel geschrieben.

Vierter Schritt

Der Inhalt des Ausnahmevektors wird in den Programmzähler geladen. Der Prozessor nimmt nun die Ausführung der durch den PC adressierten Befehle wieder auf.

Bevor wir die Ausnahmen gemäß der Tabelle in Abb. 3.5 klassifizieren, erinnern wir uns daran, daß die Prioritätsfrage entschieden wird, wenn zwei Ausnahmen gleichzeitig auftreten, um zu bestimmen, welche zuerst bearbeitet werden muß.

In Abb. 3.5 sind die verschiedenen Ausnahmegruppen dargestellt.

Priorität der Gruppen	Gruppen	Ausnahme	Priorität im Inneren einer Gruppe	Augenblick, in dem die Ausnahme anerkannt wird	Augenblick, in dem die Verarbeitung der Ausnahme beginnt (dieser kann gegebenenfalls durch $\overline{HALT}$, $\overline{BR}$ verzögert werden)
	0	RESET Busfehler Adressfehler		Ende des Taktzyklus. Der laufende Befehl wird abgebrochen	Nächster Taktzyklus
	1	TRACE Unterbrechung illegaler Bef. Bef. nicht implementiert privilegierter Befehl		Ende der Ausführung des laufendenen Befehls Ende des Buszyklus Laufender Befehl abgebrochen	Nach dem laufenden Befehl und vor dem nächsten Befehl
	2	TRAP TRAPV CHK Division durch Null	Keine Priorität, da jeder Befehl separat ausgeführt wird	Während der Abarbeitung des Befehls, nach seiner Decodierung	Nach dem Ende des laufenden Befehlszyklus

Abb. 3.5: Die verschiedenen Gruppen von Ausnahmezuständen

Wir betrachten zwei Beispiele:

- Ein Busfehler tritt während eines TRAP-Befehls auf: Der Busfehler wird vorrangig behandelt, und die Ausführung des TRAP-Befehls wird abgebrochen.
- Eine Unterbrechungsanforderung trifft während der Ausführung eines Befehls ein, und T ist gesetzt: TRACE ist vorrangig und wird zuerst, vor der Unterbrechung, berücksichtigt. Die Unterbrechung kommt dann zum Zuge, bevor die weiteren Befehle abgearbeitet werden.

Der detaillierte Ablaufplan zur Behandlung von Ausnahmen (Abb. 3.6) zeigt alle Schritte auf, die der Prozessor durchlaufen muß, bevor er weiß, welche Ausnahmeprozedur er ausführen muß.

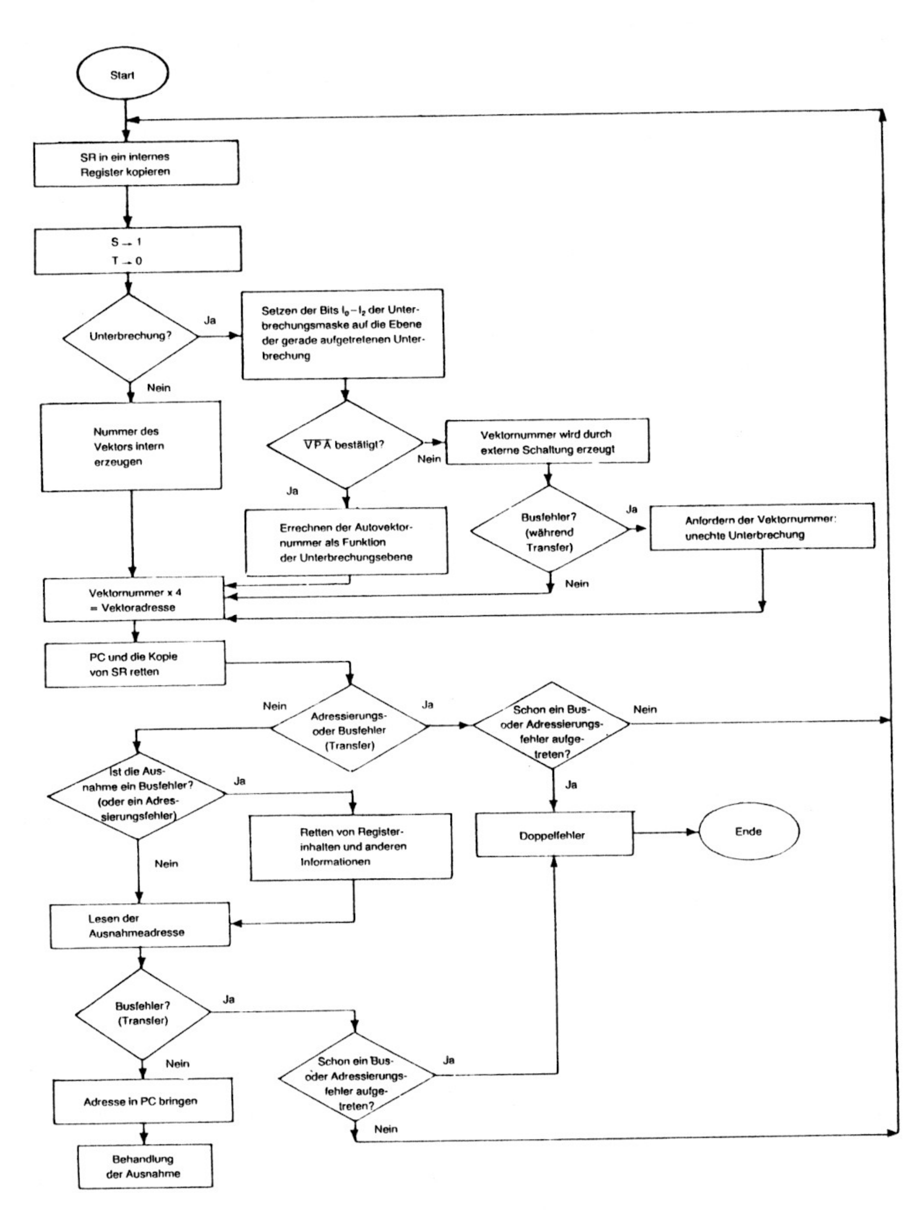


Abb. 3.6: Detaillierter Ablaufplan für die Verarbeitung von Ausnahmen (außer RESET)

BEARBEITUNG DER UNTERSCHIEDLICHEN AUSNAHMETYPEN

Die verschiedenen Ausnahmen, die in diesem Kapitel behandelt werden sollen, sind die folgenden:

1. Unterbrechungen
2. RESET
3. Befehle des Typs TRAP
4. Illegaler oder nicht implementierter Befehl
5. Privilegverletzung
6. TRACE
7. Busfehler
8. Adreßfehler
9. Unechte Unterbrechung

UNTERBRECHUNGEN

Unterbrechungsanforderungen werden über die Unterbrechungs-Steuerleitungen $\overline{IPL0}$, $\overline{IPL1}$ und $\overline{IPL2}$ geschickt, die 7 Prioritätsebenen zur Verfügung stellen.

Die externen Bausteine können mit Hilfe eines Prioritätscodierers verkettet werden, damit einer großen Anzahl von Schaltungen ermöglicht werden kann, den Prozessor zu unterbrechen (Abb. 3.7).

Die Unterbrechungsanforderung wird also durch die codierte Unterbrechungsebene auf den Leitungen $\overline{IPL0} - \overline{IPL2}$ realisiert.

Achtung: $\overline{IPL0}$, $\overline{IPL1}$ und $\overline{IPL2}$ sind low-aktiv, während das bei I0, I1 und I2 des Statusregisters genau umgekehrt ist.

Es gibt 7 Prioritätsebenen:

Die Ebene 7 hat die höchste Priorität, ist nicht maskierbar und für die Systeminitialisierung und den Wiederanlauf bestimmt. Das bedeutet auch, daß jede Unterbrechung der Ebene 7 von einer Unterbrechung der Ebene 7 unterbrochen werden kann.

Ebene 0 bedeutet, daß niemand eine Unterbrechungsanforderung gestellt hat.

Die Ebenen 1 bis 6 sind maskierbar (Abb. 3.8). Wenn eine Unterbrechung erkannt wird, wird ihre Prioritätsebene mit der Unterbrechungsmaske im Statusregister verglichen. Ist die Priorität gleich oder kleiner als

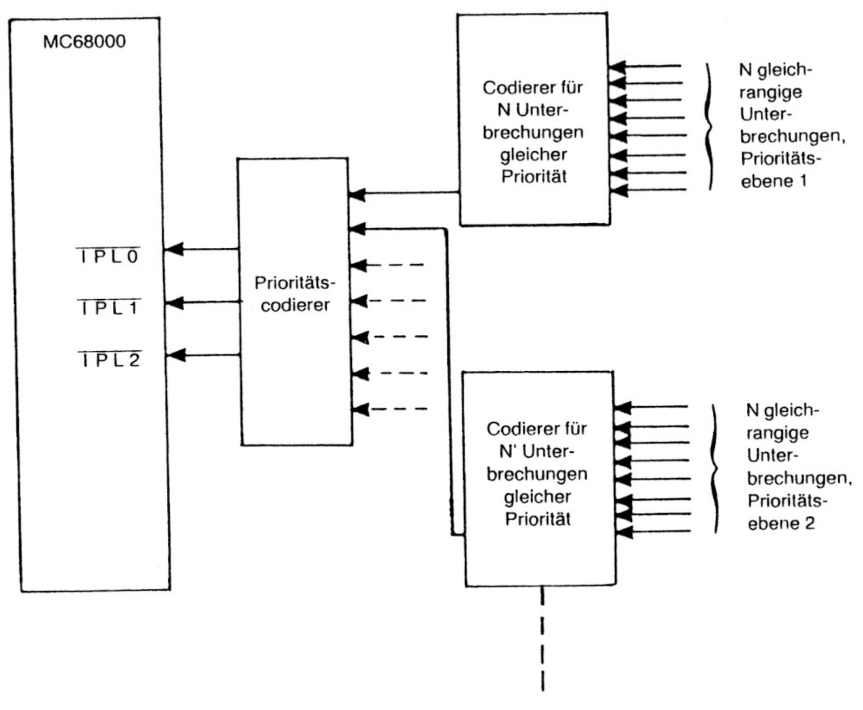


Abb. 3.7: Verkettung von Unterbrechungen

die der Maske, wird die Unterbrechung verhindert. Auf diese Weise wird also die Verarbeitung mit dem nachfolgenden Befehl fortgesetzt. Der Unterbrechungsanforderung wird nicht stattgegeben. Sie könnte ausgeführt werden, wenn der Wert der Maske im folgenden niedriger als der der Unterbrechungsebene ist und die Anforderung immer noch vorliegt. Die Unterbrechungsanforderungen, die an den Prozessor gesendet werden, verlangen also keine sofortige Bearbeitung. Vor der Übernahme wird die Unterbrechung als „schwebend“ bezeichnet (pending interrupt).

Wenn die Priorität der wartenden Unterbrechung höher als die der momentan ablaufenden ist, ist die Verarbeitungssequenz die folgende:

- Kopie des Statusregisters.
- Setzen von S auf 1.
- Setzen von T auf 0.
- Die Maske des Statusregisters wird auf die Ebene der übernommenen Unterbrechung gesetzt.

– Der Prozessor ermittelt die Vektornummer, geht in die Phase der Unterbrechungsanerkennung über und gibt die Prioritätsebene der Unterbrechung auf die Adreßleitungen A1, A2 und A3 aus. Zwei Fälle können auftreten:

Erzeugen einer Autovektornummer

Ein externer Baustein sendet das Signal $\overline{VPA}$, was den Prozessor veranlaßt, eine Autovektornummer zu erzeugen. Er erzeugt intern und unabhängig von der Prioritätsebene der Unterbrechung eine Vektornummer.

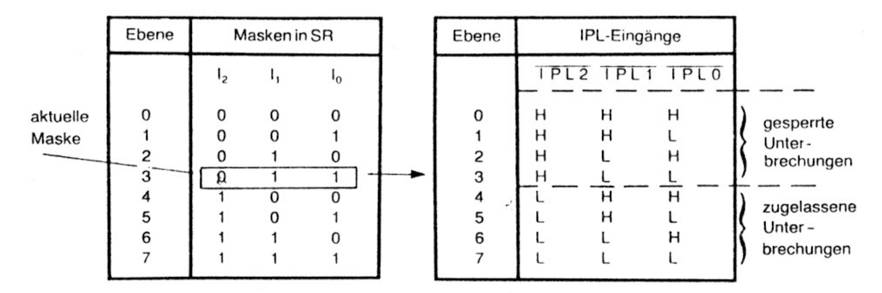


Abb. 3.8: Beziehung zwischen den Unterbrechungsmasken und den Unterbrechungseingängen

Die nicht-autovektorielle Unterbrechung

Eine externe Logik liefert die Nummer des Vektors, legt sie auf den Datenbus und beendet die Übertragung, indem sie $\overline{DTACK}$ bestätigt.

Im zweiten Fall, wo die externe Logik einen Busfehler meldet, wird der Prozessor die Vektornummer erzeugen.

– Danach werden PC und SR auf den Überwacherstapel gerettet.
– Schließlich wird der Programmzähler mit dem Inhalt der berechneten Vektoradresse geladen und die entsprechende Programmroutine ausgeführt.

Im Ablaufdiagramm der Abb. 3.9 wird der Verarbeitungsprozeß einer Unterbrechung zusammengefaßt.

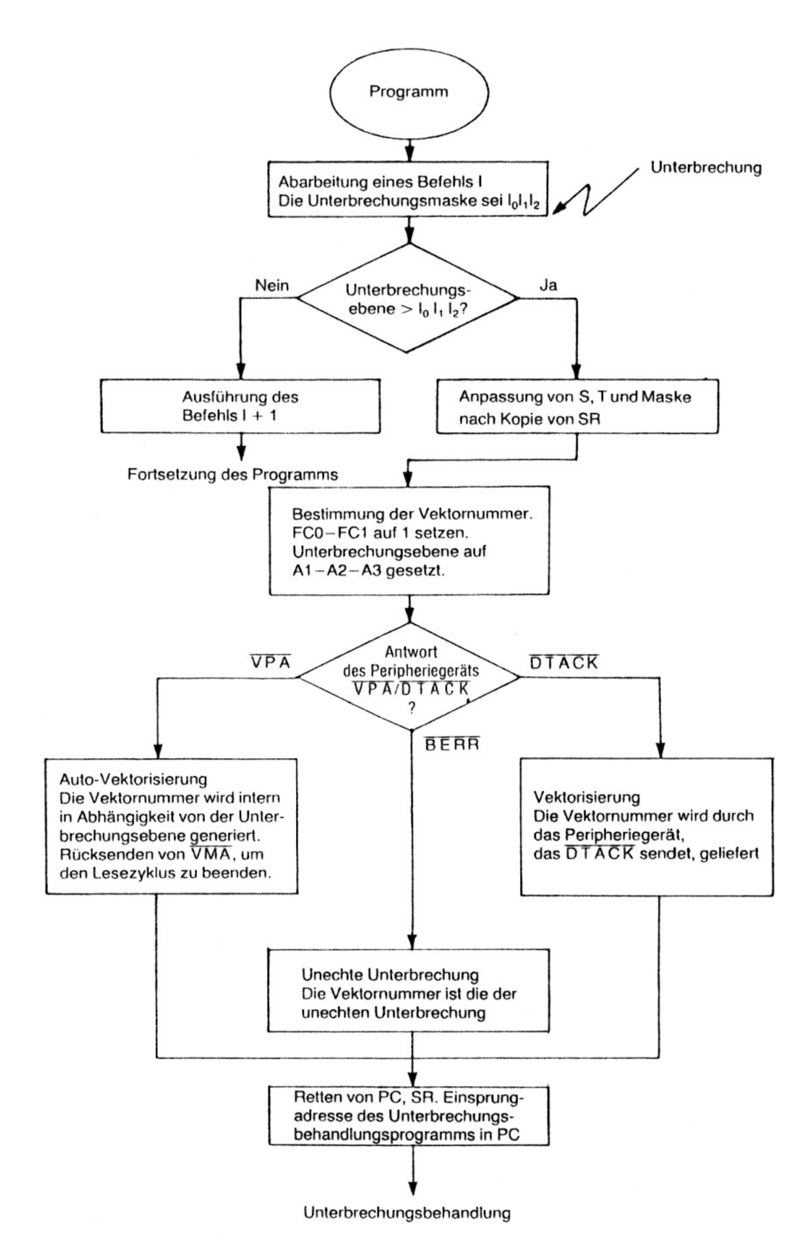


Abb. 3.9: Ablaufdiagramm für das Auftreten einer Unterbrechung

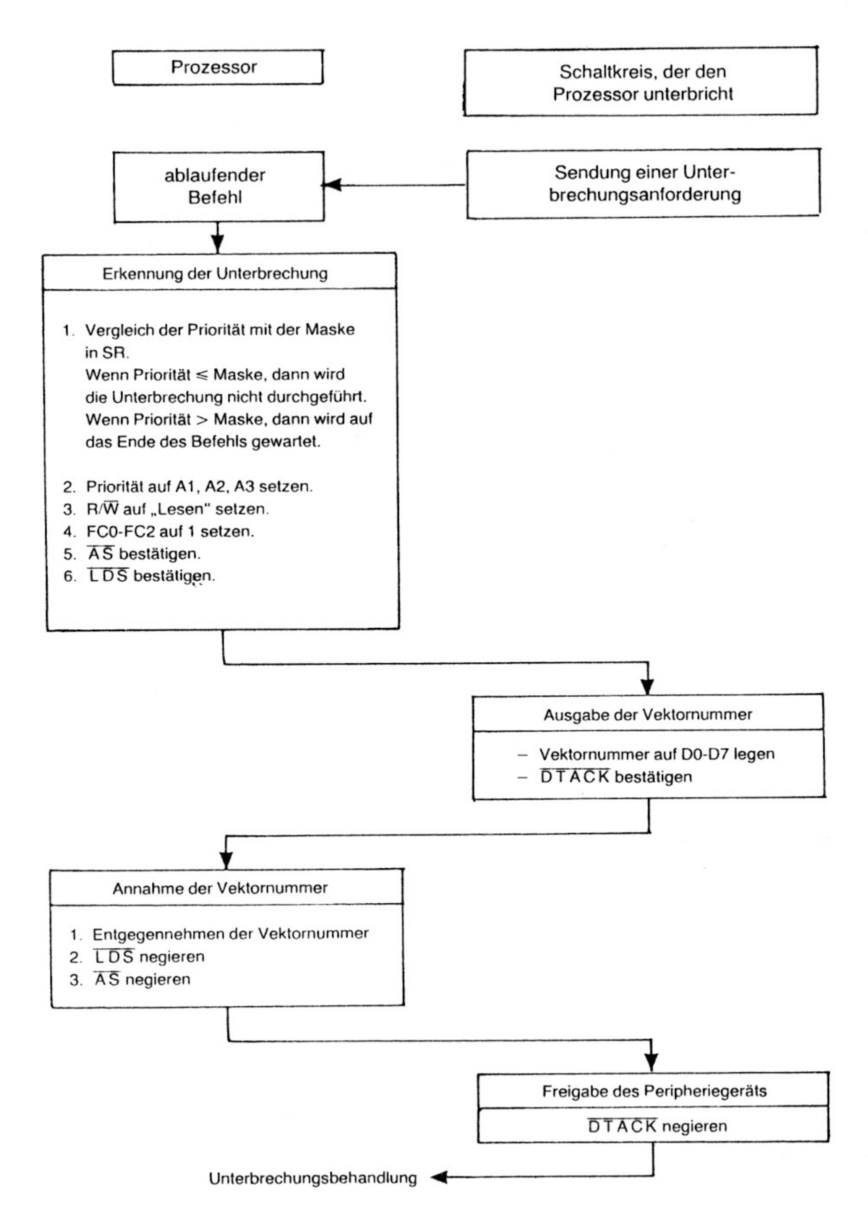


Abb. 3.10: Schemadiagramm für das Auftreten einer nicht-autovektoriellen Unterbrechung

Der Fall der nicht-automatischen Vektorisierung

Dieser Unterbrechungstyp wird durch einen Dialog zwischen dem Baustein als Verursacher der Unterbrechung und dem Prozessor charakterisiert, wie es das Ablaufdiagramm der Abb. 3.10 und das Zeitdiagramm der Abb. 3.11 zeigen.

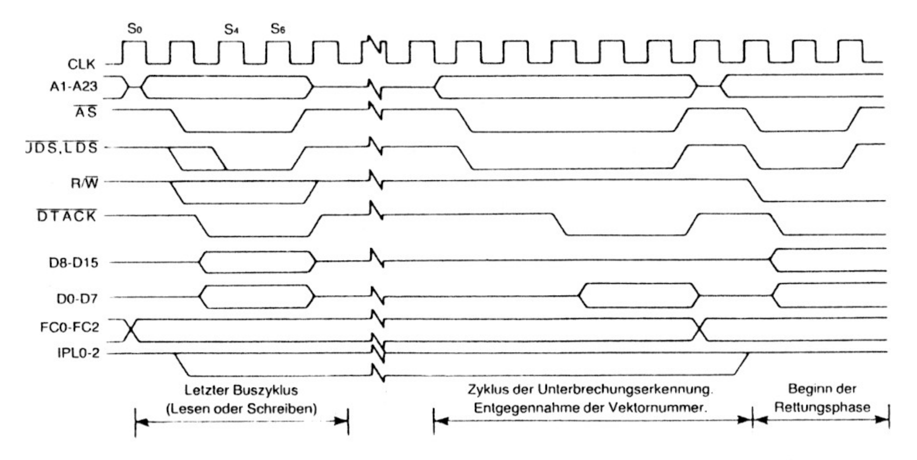


Abb. 3.11: Zeitdiagramm für das Auftreten einer nicht-autovektoriellen Unterbrechung

Die Verarbeitung der 192 Anwender-Unterbrechungen

Für den Fall der Anwender-Unterbrechungen stehen 192 Unterbrechungsvektoren zur Verfügung.

Auf diese Weise kann für jede Zahl unter 193 eine Vektornummer zugeordnet werden, die gleichzeitig die Priorität festlegen kann.

Die Vektoren sind von 64 bis 255 durchnumeriert. Die 192 Prioritätsebenen können in 8 Bits codiert werden, z. B. die Vektornummer 64 mit der niedrigsten (00000000) und die Vektornummer 255 mit der höchsten Priorität (10111111). Von dieser Prioritätsschreibweise ausgehend kann man die Vektornummmer durch Addition der Zahl 64 zur jeweiligen Prioritätsnummer erreichen.

In dem Blockdiagramm mit der Generierungsschaltung für Vektornummern, Abb. 3.12, werden zwei Speicherelemente L1 und L2 verwendet.

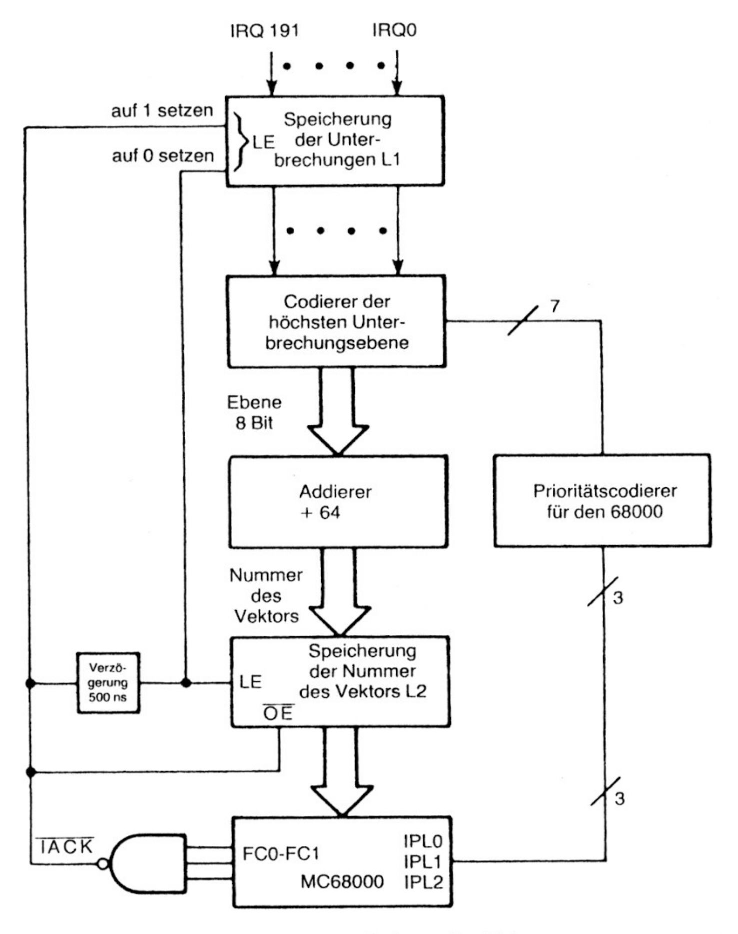


Abb. 3.12: Generierungsschaltung für Vektornummern

- L1 unterbindet jede weitere Unterbrechung, bevor nicht L2 die aktuelle Vektornummer gespeichert hat.
- L2 entnimmt dem Datenbus die Vektornummer und speichert sie, bis die Unterbrechungsquittung $\overline{IACK}$ erfolgt ist, die besagt, daß der 68000 die Unterbrechung anerkannt hat.

Nach einer genügend langen Zeit, die für die Speicherung der Vektornummer ausreicht, läßt L1 wieder neue Unterbrechungen zu.

Die autovektorielle Unterbrechung

Der Ablauf der autovektoriellen Unterbrechung ist in dem Ablaufdiagramm der Abb. 3.13 und dem Zeitdiagramm in Abb. 3.14 dargestellt.

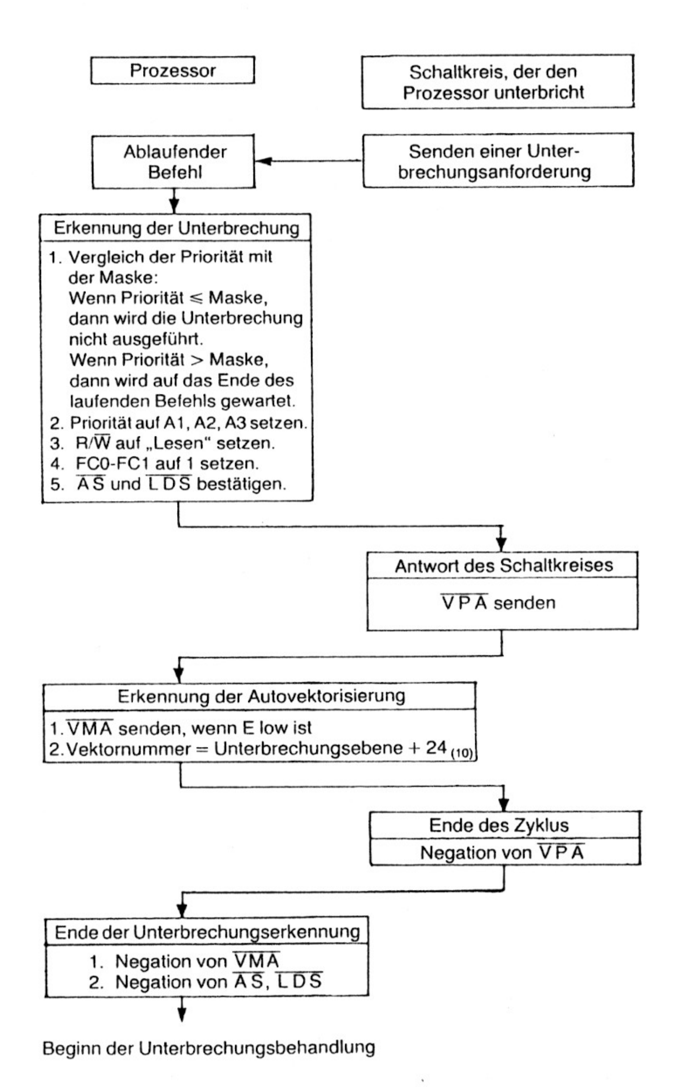


Abb. 3.13: Ablaufdiagramm für das Auftreten einer autovektoriellen Unterbrechung

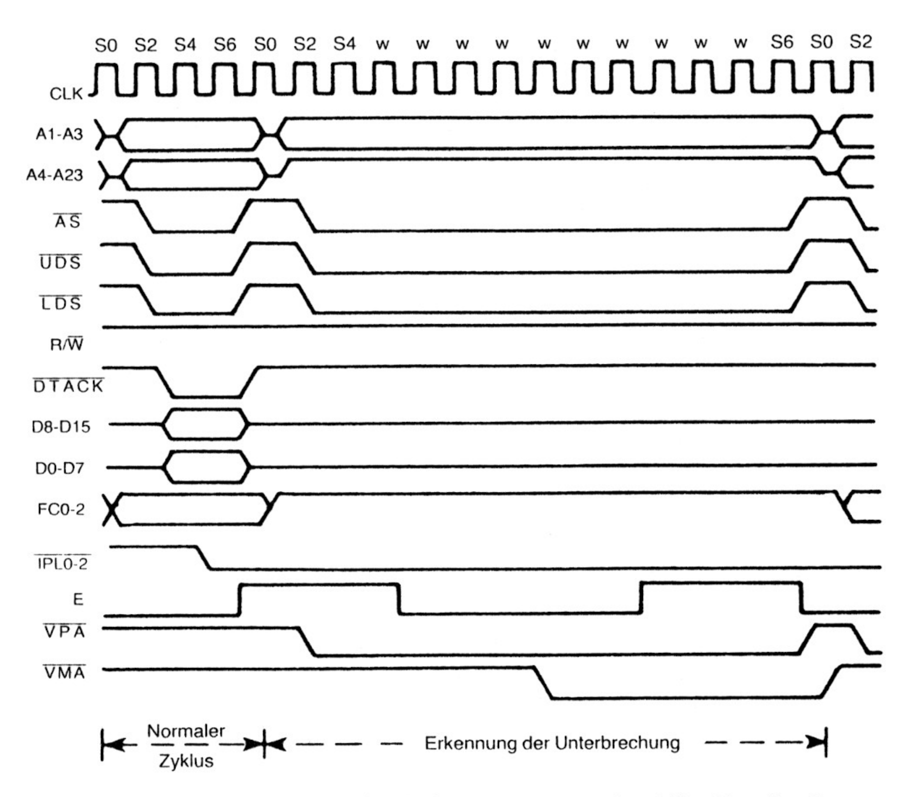


Abb. 3.14: Zeitdiagramm für das Auftreten einer autovektoriellen Unterbrechung

Das vereinfachte Schema in Abb. 3.15 zeigt die Schaltung für eine autovektorielle Unterbrechung zwischen dem PIA 6821 und dem 68000.

1. Der MC68000 erhält eine Unterbrechungsanforderung $\overline{IRQA}$ oder $\overline{IRQB}$ über die Unterbrechungssteuerleitungen $\overline{IPL0}$, $\overline{IPL1}$ und $\overline{IPL2}$.
2. Der MC68000 adressiert den 6821 und setzt sich selbst auf Erkennung der Unterbrechung: $\overline{AS}=0$, FC0=FC1=FC2=1
3. Der Prozessor empfängt $\overline{VPA}$.
4. Er erzeugt intern die Vektornummer.

Eine Erweiterung des Prinzips stellt das Schaltschema in Abb. 3.16 dar. Mehrere PIAs (6821) und ein ACIA (6850) unterbrechen den 68000.

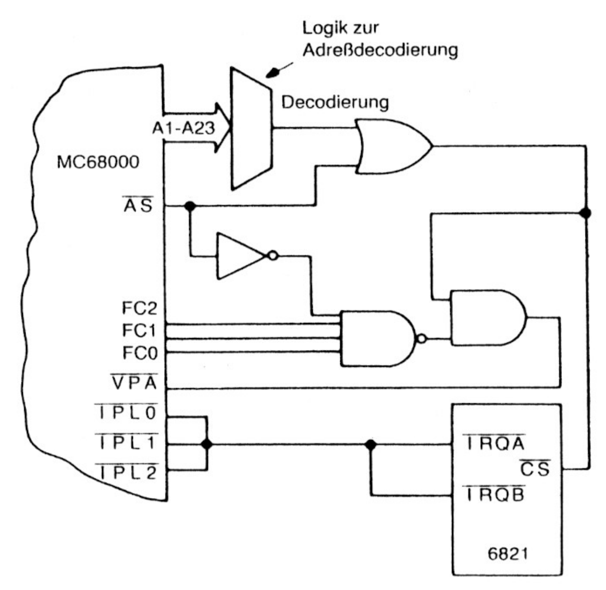


Abb. 3.15: Vereinfachtes Schaltschema für eine autovektorielle Unterbrechung des MC68000 mit dem PIA 6821

Schließlich verbinden wir die zwei Unterbrechungsarten in dem Schema der Abb. 3.17 miteinander.

RÜCKSETZUNG (RESET)

Der Anschluß RESET ist bidirektional.

Als Eingang:

Beim Einschalten des Stroms wird eine generelle Rücksetzung des Systems (Prozessor und externe Schaltungen) erzielt, indem während 100 ms $\overline{RESET}$ und $\overline{HALT}$ am 68000 angelegt werden.

Der Prozessor liest dann ab Adresse \$000000 in der Vektortabelle die entsprechende Vektornummer nach. Der Inhalt dieser Adresse wird in den Überwacherstapel geladen. Dieser ist nun initialisiert.

Danach liest der Mikroprozessor automatisch den Inhalt der Adresse \$000004 und legt ihn im Programmzähler PC ab. Der Prozessor kennt nun die Adresse des ersten auszuführenden Befehls.

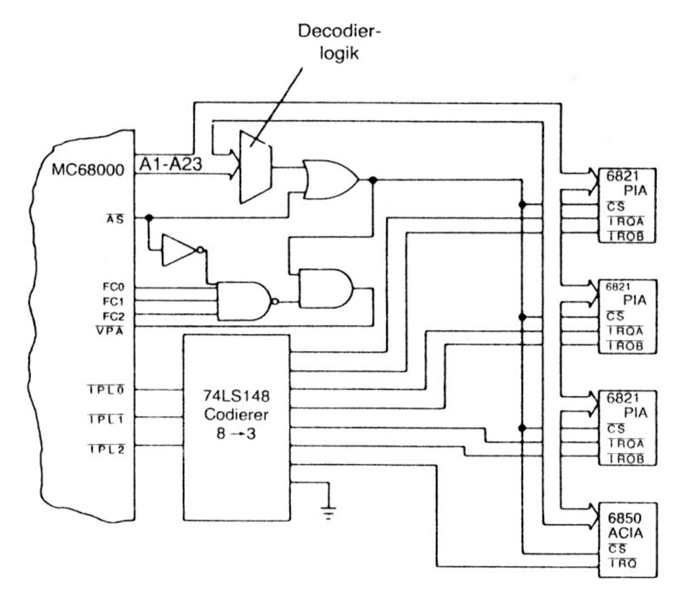


Abb. 3.16: Mehrere PIAs und ein ACIA können den MC68000 autovektoriell unterbrechen

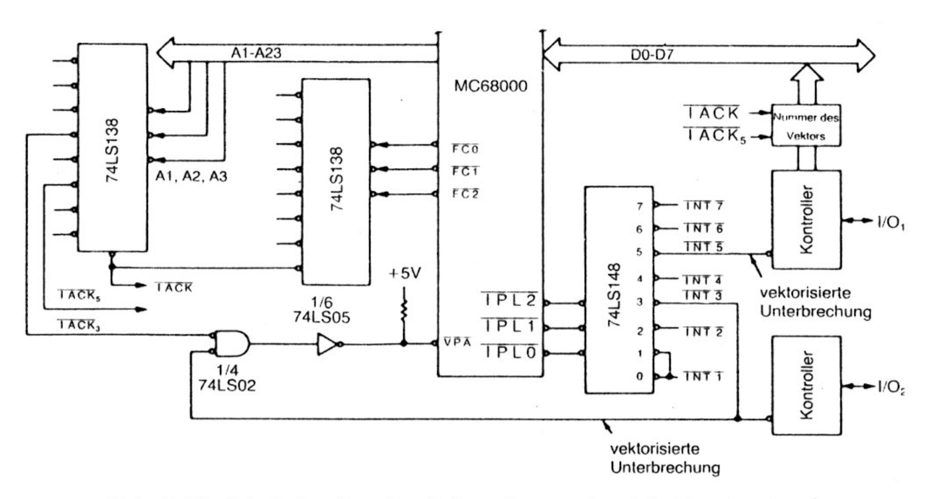


Abb. 3.17: Schaltplan für vektorielle und autovektorielle Unterbrechung

Schließlich setzt er die Unterbrechungsmaske des Statusregisters auf die höchste Ebene (Level 7).

RESET ist für den Wiederanlauf des Systems nach schwerwiegenden Fehlern bestimmt. Während eines RESET wird jede laufende Verarbei-

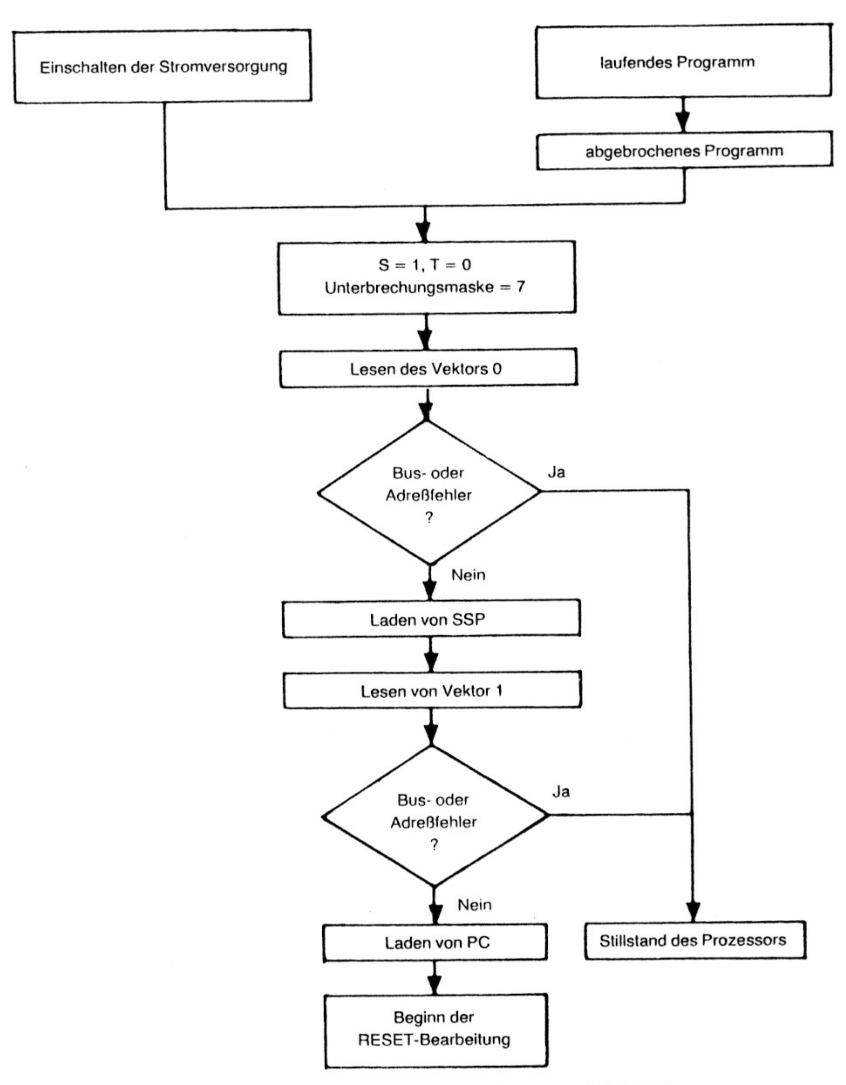


Abb. 3.18: Ablauf einer Rücksetzung (RESET)

tung vorzeitig abgebrochen und kann auch später nicht fortgeführt werden. Der Prozessor fällt in den Überwachungsmodus, und der TRACE-Modus wird ausgeschaltet.

Die Vektornummer 0 wird intern erzeugt.

Da keine Angaben über Inhalt der Register, insbesondere über den Zeiger des Überwacherstapels und den Programmzähler gemacht werden können, wird auch keine Hilfsroutine gestartet.

Tritt während der Initialisierungsphase ein Bus- oder Adreßfehler auf, kommt der Prozessor zum Stillstand, und die gesamte Verarbeitung hört auf. Nur ein erneutes RESET kann das System wieder starten.

Diese RESET-Sequenz wird im Ablaufdiagramm der Abb. 3.18 dargestellt.

Als Ausgang:

Die Ausführung des Befehls RESET bewirkt das Setzen des RESET-Signals für 124 Taktimpulse. Bei diesem Vorgehen wird das System durch den Prozessor selbst initialisiert.

Auf diese Weise wird keine Wirkung auf den Zustand des Prozessors ausgeübt; die Register bleiben unverändert. Nach beendeter Ausführung dieses Befehls wird der nächste Befehl bearbeitet.

BEFEHLE DES TYPS TRAP

Befehle des Typs TRAP können Ausnahmezustände veranlassen. Sie treten entweder bei Erkennen eines abnormalen Programmablaufs auf oder nach Ausführen von Befehlen, die die Aufgabe haben, einen Ausnahmezustand zu erzeugen.

Der TRAP-Befehl

Dieser Befehl erzwingt immer eine Ausnahme. 16 Vektoren sind für den TRAP-Befehl reserviert. Die Vektornummer wird intern erzeugt und stimmt mit dem Anfang der Tabelle überein, die die 16 verfügbaren Vektoren beinhaltet. Die genaue Nummer wird im Befehl (TRAP 0 bis TRAP 15) angegeben. Die Ausnahmeverarbeitung ist die gleiche wie bei Unterbrechungen. TRAP wird hauptsächlich für den Aufruf des Betriebssystems vom Anwenderprogramm aus verwendet. Das Prinzip wird in Abb. 3.19 veranschaulicht.

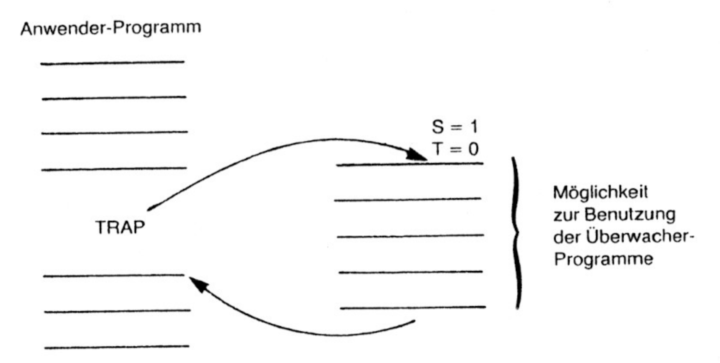


Abb. 3.19: Prinzipielle Arbeitsweise des TRAP-Befehls

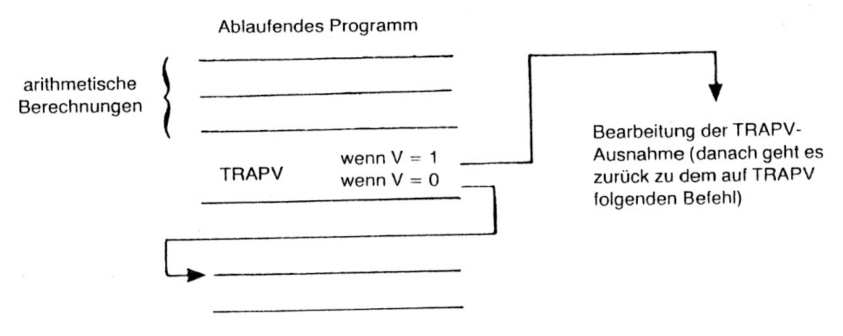


Abb. 3.20: Prinzipielle Arbeitsweise des TRAPV-Befehls

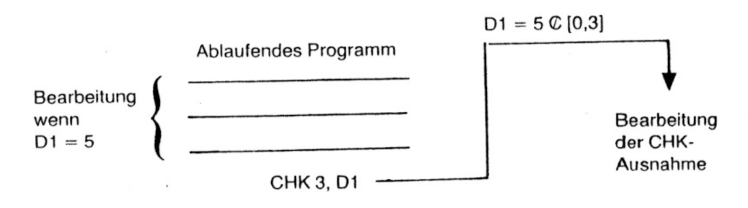


Abb. 3.21: Prinzipielle Arbeitsweise des CHK-Befehls

Der TRAPV-Befehl

Der TRAPV-Befehl löst eine Ausnahmeverarbeitung nach dem TRAPV-Vektor aus, wenn die Voraussetzung für die Kapazitätsüberschreitung (overflow) eines Operanden erfüllt ist. Das Verarbeitungsschema sehen Sie in Abb. 3.20.

Der CHK-Befehl

Der CHK-Befehl überprüft, ob sich das untere Wort eines spezifizierten Datenregisters in dem Intervall 0 bis obere Grenze befindet.

Ist der Registerinhalt kleiner als 0 oder höher als die obere Grenze, wird eine Ausnahmeverarbeitung eingeleitet.

Der DIVS- und DIVU-Befehl

Die Befehle DIVS (dividiere mit Vorzeichen) und DIVU (dividiere ohne Vorzeichen) erzwingen eine Ausnahme, wenn eine Division durch Null beabsichtigt wird.

ILLEGALE UND NICHTIMPLEMENTIERTE BEFEHLE

Illegaler Befehl

Der Ausdruck illegaler Befehl bezeichnet ein Binärmuster, dessen Operationsteil nicht einer legalen Bitkombination für ein Befehlswort entspricht. Wenn ein solcher Befehl zur Ausführung gelangen soll, wird die Ausnahmebedingung „illegaler Befehl" signalisiert.

Nichtimplementierter Befehl

Motorola hat nur ungefähr 80% der möglichen Bitkombinationen für die Implementierung von Befehlswörtern verwendet. Zu den unbenutzten Binärcodes gehören die Binärmuster 1010 und 1111 an den Stellen 12 bis 15, also die vier höchstwertigen Bits, die den Operationscode für einen Befehl darstellen.

Auf diese Weise kann die Verwendung nichtimplementierter Befehle softwaremäßig entdeckt werden, oder sie können bewußt für Emulationszwecke eingesetzt werden.

Die Verarbeitung von Ausnahmen entspricht der von Unterbrechungen, bei denen die Vektornummer intern erzeugt wird. Für die nichtimplementierten Operationscodes 1010 und 1111 wird die Vektornummer 10 bzw. 11 verwendet und für die illegalen Befehle die Vektornummer 4.

NICHTAUTORISIERTE VERWENDUNG PRIVILEGIERTER BEFEHLE

Um die Sicherheit eines DV-Systems zu garantieren, sind einige Befehle privilegiert. Sie können nur im Supervisor-Modus verwendet werden, und jeder Versuch, sie im Anwender-Modus zu benutzen, führt zu dem Ausnahmezustand „Privilegverletzung". Die privilegierten Befehle sind:

STOP
RESET
RTE
MOVE to SR
ANDI to SR
EORI to SR
ORI to SR
MOVE USP

Die Vektornummer, die bei einem Ausnahmezustand des Typs Privilegverletzung erzeugt wird, ist 8.

DIE BETRIEBSART TRACE

Diese Art der Befehlsausführung erlaubt eine Unterstützung der Programmentwicklung und Fehlerbehebung, indem die Befehle Schritt für Schritt ausgeführt werden. Nach jedem Befehl wird ein Ausnahmezustand veranlaßt, wenn vor der Ausführung des Befehls festgestellt wird, daß das T-Bit des Statusregisters gesetzt ist.

Im folgenden werden verschiedene Fälle betrachtet, in denen Ausnahmezustände eintreten, während der TRACE-Modus aktiviert ist.

1. Fall:

Der betreffende Befehl wurde nicht ausgeführt oder ist aus einem der folgenden Gründe abgebrochen worden:

- Eine Unterbrechungsanforderung ist erkannt worden.
- Der Befehl ist ungültig oder privilegiert.

- RESET
- Busfehler
- Adreßfehler

Unter diesen Umständen kommt TRACE nicht zur Ausführung.

2. Fall:

Der Befehl ist gerade in der Ausführung begriffen, und eine Ausnahme wird durch den Befehl herbeigeführt. Dann wird zuerst die Ausnahme abgearbeitet und danach der TRACE-Befehl.

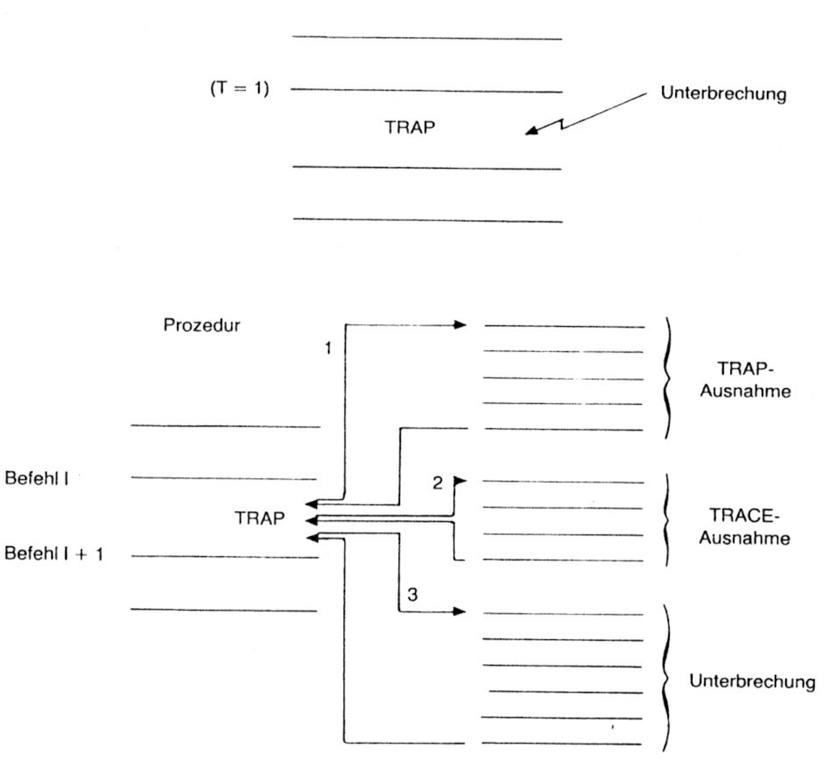


Abb. 3.22: Aufeinanderfolge von mehreren Ausnahmen

Im Beispiel der Abb. 3.22 wird zunächst der TRACE-Modus aktiviert, und dann wird ein TRAP-Befehl ausgeführt, währenddessen eine Unterbrechung auftritt.

BUSFEHLER

Der Busfehler ist ein extern erzeugtes Eingangssignal, das einen beim Datentransfer aufgetretenen Fehler anzeigt, der über eine Ausnahmeprozedur abgefangen werden soll.

Beim Empfang von $\overline{BERR}$ wird der gerade ablaufende Buszyklus abgebrochen. Ein in der Ausführung begriffener Befehl wird unterbrochen, um die Ausnahmesequenz einzuleiten, deren Ablauf teilweise schon bekannt ist:

- Kopie von SR
- S=1
- T=0
- Interne Generierung der Vektornummer
- Da der Prozessor den laufenden Befehl nicht beenden konnte, ist die Rettungsroutine komplexer und wird daher im einzelnen angegeben:
  - Speichern von PC, SR. Der gerettete Wert von PC wird um 2 bis 10 Bytes, ausgehend von der Adresse des ersten Wortes des Befehls, bei dem der Busfehler hervorgerufen wurde, erhöht (Vorgriffphänomen).
  - Sicherung des Befehlsregisters.
  - Sicherung der Adresse, für die die Anomalie aufgetreten ist.
  - Speichern des Zustands von R/$\overline{W}$, um festzuhalten, ob sich der Prozessor in einem Lese- oder Schreibvorgang befunden hat.
  - Speichern des Zustands I/N, um anzugeben, ob der Prozessor gerade im Begriff war, einen Befehl auszuführen.
    D. h.:
    I/N=0, normaler Befehl oder Ausnahme der Gruppe 2 (siehe Tabelle 3.23).
    I/N=1, der Prozessor verarbeitete eine Ausnahme der Gruppe 0 oder 1 (siehe Abb. 3.23).
  - Speichern der Funktionscodes FC0, FC1 und FC2.

Mit Hilfe dieser Informationen ist es möglich, bei Auftreten eines Fehlers eine Software-Diagnose zu stellen. Tritt während der Verarbeitung einer Ausnahme ein weiterer Fehler auf, gibt es einen Doppelfehler, und der Prozessor kommt zum Stillstand. Dann kann nur RESET ihn neu starten.

Gruppe	Ausnahme	Ausnahmeverarbeitung beginnt:
0	Rücksetzung Busfehler Illegaler Adresse	Am Ende eines Taktzyklus
1	TRACE, Unterbrechung Illegaler Befehl.	Am Ende eines Befehlszyklus
	Nichtvorhandener Befehl, Privilegverletzung.	Am Ende eines Buszyklus
2	TRAP, TRAPV, CHK, Division durch 0	Am Ende eines Befehlszyklus

Abb. 3.23: Ausnahmegruppierung und Priorität

ADRESSFEHLER

Die Adreßfehler treten auf, wenn der Prozessor versucht, über eine ungerade Adresse einen Wort- oder Doppelwortoperanden anzusprechen.

Unter diesen Umständen wird der Buszyklus abgebrochen, und eine Ausnahmeverarbeitung wird durchgeführt, die der bei einem Busfehler ähnelt.

Abb. 3.24 zeigt eine Übersicht über Sicherungsvorgänge bei Auftreten eines Bus- oder Adreßfehlers.

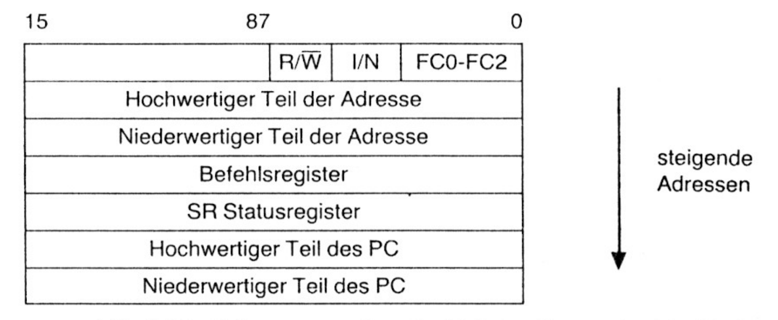


Abb. 3.24: Sicherungsvorgänge im Fall eines Bus- oder Adreßfehlers

DIE NICHTINITIALISIERTE UNTERBRECHUNG

Im Fall der Unterbrechung durch ein nichtinitialisiertes Peripheriegerät wird die Vektornummer 15 angenommen. So erzwingt beispielsweise ein RESET der Peripherie-Bausteine der 68000-Familie die Vektornummer 15. Auf diese Weise wird die mangelnde Fähigkeit dieser Peripherie-Bausteine ausgeglichen.

Kapitel 4

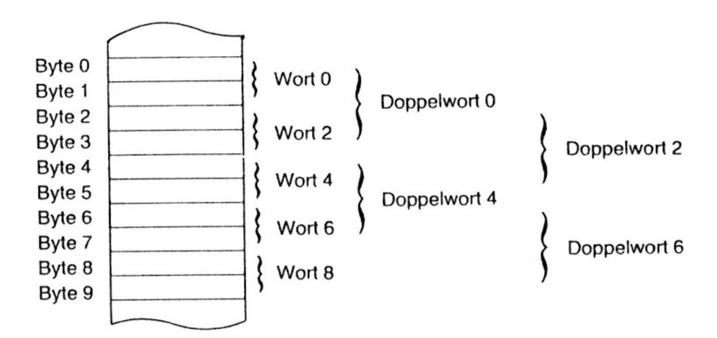
Die Speicherverwaltung

DATENORGANISATION IM SPEICHER UND DIE BEDINGUNGSCODES

Zunächst wollen wir uns einmal die Organisation der Daten im Speicher vor Augen führen, damit wir die Begriffe „gerade“ und „ungerade“ Adressierung verstehen können. Der Speicher kann in 16-Bit-Einheiten dargestellt werden.

Wort Byte 000000	000000 Byte 000001
Wort Byte 000002	000002 Byte 000003
Wort Byte FFFFFE	FFFFFE Byte FFFFFF

Die Begriffe Byte, Wort und Doppel- oder Langwort werden im folgenden Schema näher veranschaulicht:



Jetzt können wir folgende Präzisierung vornehmen: Eine Byte-Adresse kann gerade oder ungerade sein, eine Wort-Adresse muß gerade sein, und eine Doppelwort-Adresse muß auch gerade sein. Ein aus Worten zusammengesetzter Befehl muß darum notgedrungen auch eine gerade Adresse haben.

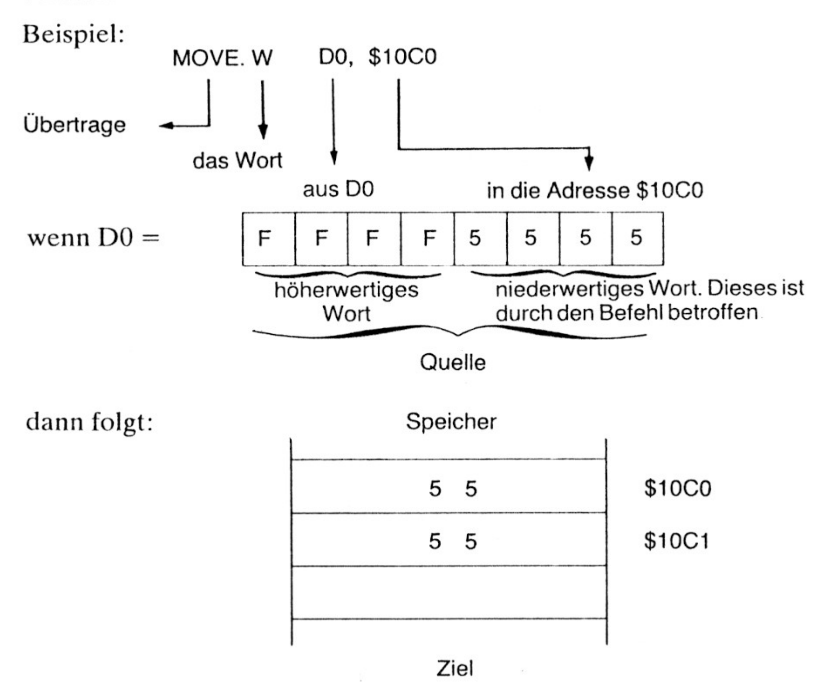
Beispiel für eine ungültige Adresse:

MOVE.W D1,$3115
(Übertrage die 16 Bit von D1 zur
Speicheradresse $3115)

Diese Operation überträgt ein Wort. Daher ist die angegebene ungerade Adresse $3115 unkorrekt.

In einem Befehl, der ein Datenregister Dn betrifft, gibt man dem Prozessor einen Datenlängencode an, um die bei dem Befehl verwendeten Datenlängen zu spezifizieren. Dabei bedeuten „.B" (Byte), „.W" (Wort) oder „.L" (Langwort), daß sich die Operation auf 8 oder 16 Bit des niederwertigen Teils des Registers oder auf die ganzen 32 Bit des Registers bezieht.

Beispiel:



Eine Verwendung dieser drei Varianten ist nicht bei allen Befehlen möglich. Insbesondere kann man für Operationen, die sich auf Adreßregister An beziehen, nicht „.B" verwenden.

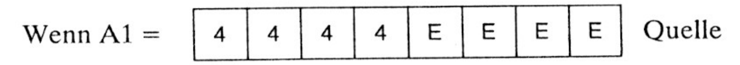
Beispiel:

MOVE.B An,$2000 ist verboten!

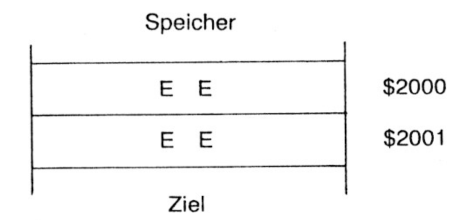
Nur die Operationen unter Verwendung von „.W" und „.L" sind erlaubt.

Beispiel: An ist der Quelloperand (Ursprungsdatenwort)

MOVE.W A1,$2000
(Übertrage das Wort von A1 zur
Adresse $2000)



dann

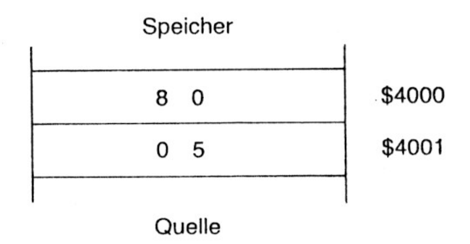


Beispiel: Wenn An der Zieloperand (Senke oder Bestimmungsoperand) ist und in der Operation „.W" spezifiziert wird, wird der Operand vor Ausführung des Befehls vorzeichenbehaftet erweitert.

Beispiel:

MOVE.W $4000,A1
(Übertrage das Wort ab der Adresse
$4000 in das Register A1)

Wenn:



dann:

A1 =	F	F	F	F	8	0	0	5	Ziel

Ein weiteres Beispiel:

```
                    00001000   ORG      $1000       : Initialisierung des
                                                    : Programms ab $1000
00001000            327CB000   MOVE.W   #$8000, A1  : Übertragung von
                                                    : $8000 nach A1.
                               END
TOTAL ERRORS        0
TOTAL WARNINGS 0

D0-D7      26534354   30303033   46495820   00000000
           20202020   20202020   00000000   00000061
A0-A7      45533420   FFFF8000   00000000   30303033
           000012BD   014E19F4   00000000   00000000
PC = 00001004 SR = 8000
```

DAS BEDINGUNGSCODEREGISTER CCR

Das Bedingungscoderegister ist im unteren Teil des Statusregisters SR enthalten (Abb. 4.1).

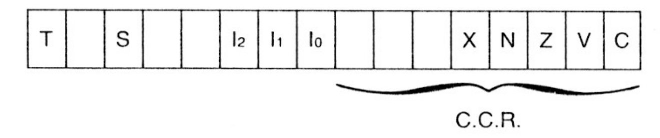


Abb. 4.1: Das Bedingungscoderegister ist ein Teil des Statusregisters

Wir wollen nun einige Beispiele von Operationen betrachten, die den Inhalt des Registers CCR verändern.

Beispiel 1:

ADD.W D0,D1
(Addiere die Worte, die in D0 und D1 enthalten sind, und schreibe das Ergebnis in D1)

Vor der Ausführung:

D0=$00218013, D1=$00038065, CCR=$00

Ausführung:

```
  8013
+8065
------
 10078
```

also N=0, Z=0, C=1, X=1, V=1

Nach der Ausführung:

D0=$00218013, D1=$00030078, CCR=$13

Der Befehl ADD setzt die Bedingungscodes gemäß dem Resultat der Operation.

Beispiel 2:

Programm:

```
          00001000      ORG      $1000
00001000  203C00218013  MOVE.L   #$00218013, D0
00001006  223C00038065  MOVE.L   #$00038065, D1
0000100C  023C00E0      ANDI     #$E0, CCR
00001010  D240          ADD.W    D0, D1
                        END
TOTAL ERRORS     0
TOTAL WARNINGS 0
```

Dieses Programm hat die Aufgabe, den 32-Bit-Wert $00218013 in das Register D0 und den 32-Bit-Wert $00038065 in das Register D1 zu laden. Dann werden die signifikanten Bits des Registers CCR auf 0 gesetzt, bevor die Addition von D0 und D1 in 16 Bits ausgeführt wird.

Programmlauf:

```
                                  MOVE.L #$00218013, D0
D0-D7    00218013  30303033  46495820  00000000
         20202020  20202020  00000000  00000061
A0-A7    45533520  434E3130  00000000  30303033
         000012BD  0162B1F1  00000000  00000000
PC = 00001006 SR = 8000          MOVE.L #$00038065, D1

D0-D7    00218013  00038065  46495820  00000000
         20202020  20202020  00000000  00000061
```

```
A0-A7     455335520  434E3130   00000000    30303033
          000012BD   0162B1F1   00000000    00000000
PC = 0000100C SR = 8000         ANDI.B #$000000E0, SR
D0-D7     00218013   00038065   46495820    00000000
          20202020   20202020   00000000    00000061
A0-A7     45535520   434E3130   00000000    30303033
          000012BD   0162B1F1   00000000    00000000
PC = 00001010 SR = 8000         ADD   D0,    D1

D0-D7     00218013   00030078   46495820    00000000
          20202020   20202020   00000000    00000061
A0-A7     45533520   434E3130   00000000    30303033
          000012BD   0162B1F1   00000000    00000000
PC = 00001012 SR = 8013      END
```

Beispiel 3:

MOVE.B #$80,D1
(Übertrage das Byte $80 nach D1)

Vor der Ausführung:

D1=$17832428 und CCR=$17 (X=1, N=0, Z=V=C=1))

Ausführung:

$80 wird anstelle von $28 in D1 geschrieben.

Nach der Ausführung:

D1=$17832480 und CCR=$18

Durch den MOVE-Befehl werden V und C auf 0 zurückgesetzt. X bleibt von der Übertragung unbeeinflußt und ist weiterhin 1. N wird auf 1 gesetzt, denn das höchstwertige Datenbit ist 1.

Beispiel 4:

NOT.L D0
(Bilde das Einerkomplement von D0)

Von der Ausführung:

D0=$02110443 und CCR=$00

Ausführung:

D0=$FDEEFBBC; das Ergebnis ist negativ, also wird N=1, und da das Ergebnis nicht 0 ist, wird Z=0 gesetzt.

Der Befehl NOT veranlaßt, daß die Bedingungscodes C und V auf 0 zurückgesetzt werden und das Bit X unverändert bleibt.

Nach der Ausführung:

D0=$FDEEFBBC und CCR=$08

Beispiel 5:

ADDA.W D0,A0
(Addition des erweiterten Wortes D0 zu A0)

Vor der Ausführung:

A0=$00002250, D0=$00008020, CCR=$17

Ausführung:

```
 FFFF8020
+00002250
---------
 FFFFA270
```

Nach der Ausführung:

A0=$FFFFA270, D0=$00008020, CCR=$17

Der Befehl ADDA läßt alle Bedingungscodes unverändert.

Abschließend kann man über das Setzen der Bedingungscodes folgendes sagen:

- Das Setzen ist von dem jeweiligen Befehl abhängig (ADD setzt die üblichen Bedingungscodes bei einer Addition, während ADDA alle Codes unverändert läßt).
- Die Codes stehen in Beziehung zur Länge der Operanden.

DIE ADRESSIERUNGSARTEN

Der Mikroprozessor MC68000 besitzt 14 unterschiedliche Adressierungsarten, die man in 6 Gruppen einteilen kann:

- Absolut
  Absolut kurz
  Absolut lang

– Register direkt
  Datenregister direkt
  Adreßregister direkt
  Statusregister direkt

– Unmittelbar
  Unmittelbar einfach
  Unmittelbar schnell

– Register indirekt
  Adreßregister indirekt
  Adreßregister indirekt mit Postinkrement
  Adreßregister indirekt mit Predekrement
  Adreßregister indirekt mit Adreßdistanz
  Adreßregister indirekt mit Index und Adreßdistanz

– Relativ zum Programmzähler
  Relativ mit Adreßdistanz
  Relativ mit Index und Adreßdistanz

Wir werden nun die verschiedenen Adressierungsarten betrachten, indem wir ihre Definition geben und sie anhand von Beispielen veranschaulichen.

ABSOLUTE ADRESSIERUNG

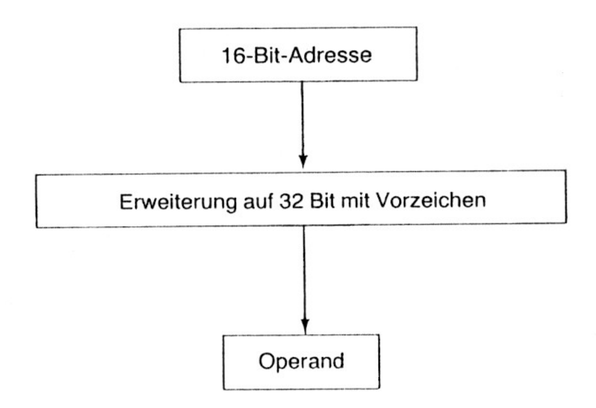
Bei diesem Adressierungstyp ist die effektive Adresse selbst als Operand spezifiziert.

– Bei der „absolut kurzen" Adressierung ist der Operand eine 16-Bit-Adresse. Sie erlaubt entweder auf die ersten 32 KByte oder auf die höchsten 32 KByte im Speicher zuzugreifen und erfordert ein Erweiterungswort.

– Bei der „absolut langen" Adressierung ist der Operand eine 32-Bit-Adresse. Sie erlaubt den Zugriff auf irgendeinen Speicherplatz des 16-MByte-Speichers des 68000 und erfordert zwei Erweiterungsworte

Absolut kurze Adressierung

Prinzipielle Arbeitsweise:

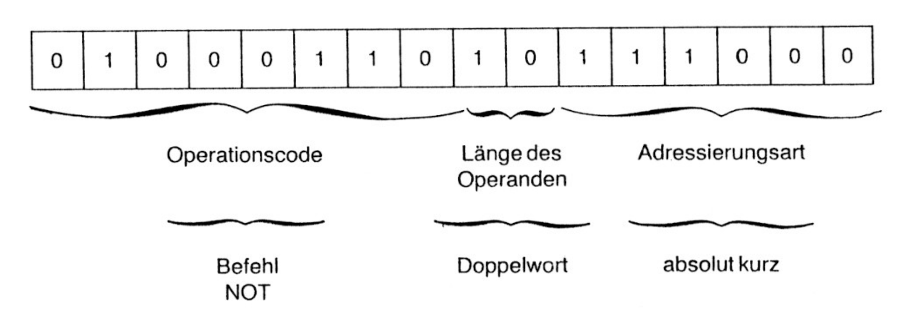


Wir wollen uns nun Beispiele für absolut kurze Adressierung ansehen.

Beispiel 1:

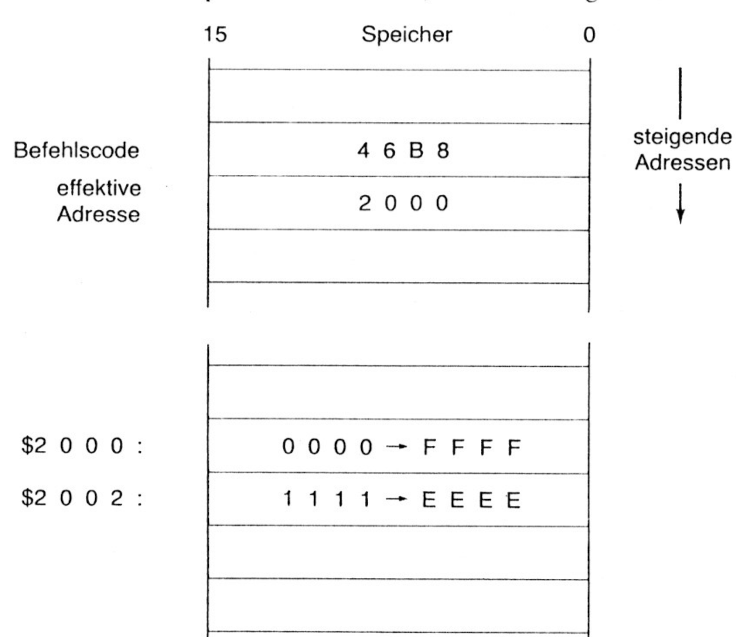
> NOT.L $2000
> (Bilde das Einerkomplement des Doppelwortes, das ab der Adresse $2000 gespeichert ist)

Befehlscode:



Der Befehlscode entspricht also dem hexadezimalen Wert $46B8.

Wenn wir uns den Speicher anschauen, sehen wir folgendes:



Beispiel 2:

MOVE.W $1000,$2000
(Übertrage das Wort von Adresse
$1000 zur Adresse $2000)

Das Format des MOVE-Befehls sieht folgendermaßen aus:

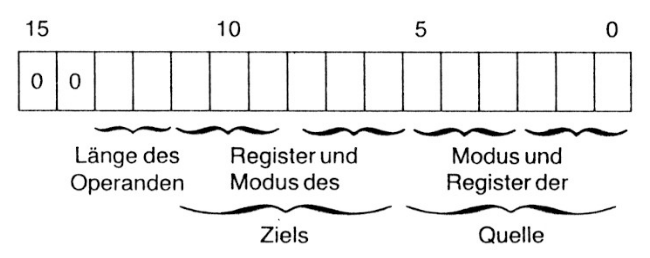


Abb. 4.2: Format des MOVE-Befehls

Um diesen Befehl als Code schreiben zu können, muß man die Tabellen für die Operationscodes und für die Adressierungsarten zu Hilfe nehmen.

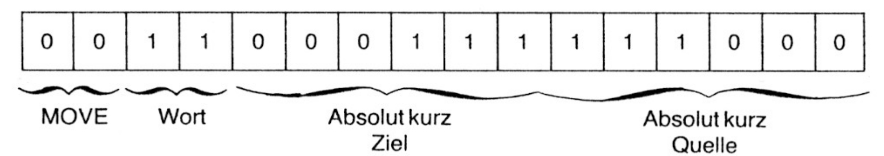
Tabelle der Operationscodes:

Bits 12 bis 15	Operation
0000	Bitmanipulation, MOVEP, unmittelbar
0001	MOVE Bytes
0010	MOVE Doppelworte
0011	MOVE Worte
0100	Andere Befehle
0101	ADDQ, SUBQ, Scc, DBcc
0110	Bcc, BSR
0111	MOVEQ
1000	OR, DIV, SBCD
1001	SUB, SUBX
1010	unbenutzt
1011	CMP, EOR
1100	AND, MUL, ABCD, EXG
1101	ADD, ADDX
1110	Schiebe- und Rotierbefehle
1111	unbenutzt

Tabelle der Adressierungsarten:

Adressierungsmodus	Modus	Register
Datenregister direkt	000	Nummer des Registers
Adreßregister direkt	001	Nummer des Registers
Adreßregister indirekt	010	Nummer des Registers
Adreßregister indirekt mit Post-inkrementation	011	Nummer des Registers
Adreßregister indirekt mit Prede-krementation	100	Nummer des Registers
Adreßregister indirekt mit Adreß-distanz	101	Nummer des Registers
Adreßregister indirekt mit Index	110	Nummer des Registers
Absolut kurz	111	000
Absolut lang	111	001
Relativ zum Programmzähler mit Adreßdistanz	111	010
Relativ zum Programmzähler mit Index	111	011
Statusregister unmittelbar	111	100

Also sieht der Befehlscode für MOVE folgendermaßen aus:



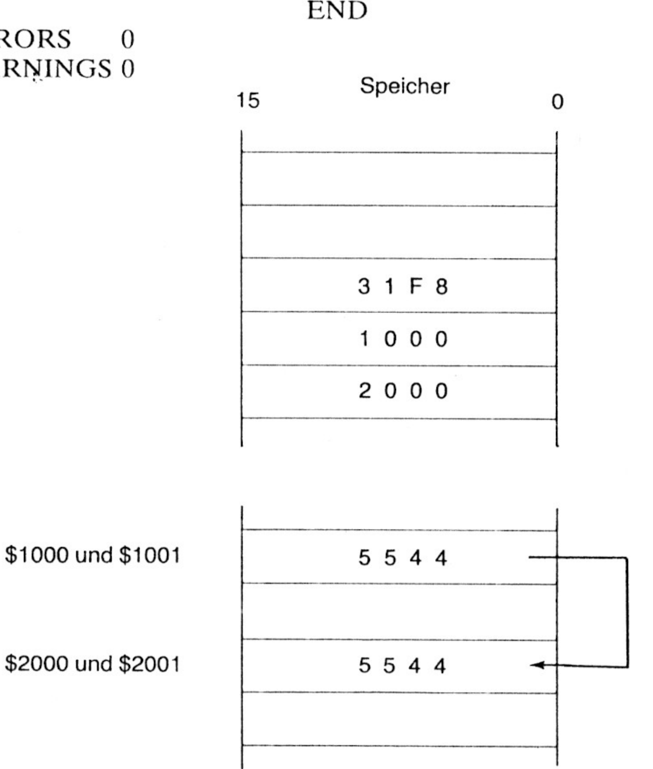
Der hexadezimale Code dieses Befehls lautet:

31F8
1000
2000

Wir finden diesen Code im folgenden Assembler-Programm wieder:

```
                00001000        ORG        $1000
00001000        31F810002000    MOVE.W     $1000, $2000
                                END

TOTAL ERRORS      0
TOTAL WARNINGS 0
```

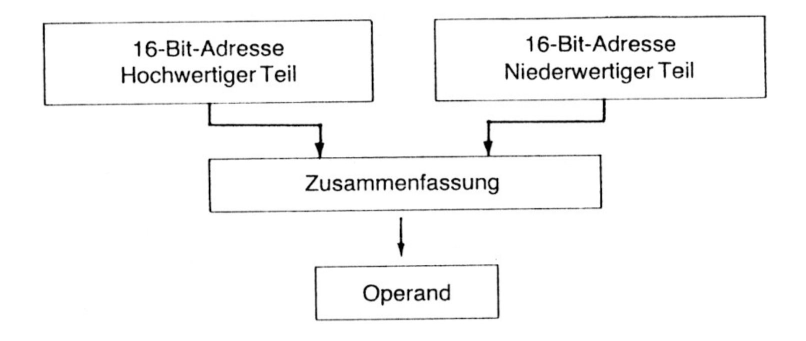


Natürlich können nur bestimmte Adreßbereiche mit absolut kurzer Adressierung beschrieben werden, da die 32-Bit-Adresse durch die vorzeichenbehaftete Erweiterung auf 32 Bits erhalten wird.

32 Bits	Darstellung der unteren 16 Bits einer 32-Bit-Adresse
00000000 00007FFF	0000 7FFF
00008000 FFFF7FFF	Darstellung als absolut kurze Adresse nicht möglich
FFFF8000 FFFFFFFF	8000 FFFF

Absolute lange Adressierung

Prinzipielle Arbeitsweise:



Beispiel für absolut lange Adressierung:

NOT.L $032000

Das Befehlsformat für den Befehl NOT zur Bildung des Einerkomplements wird in Abb. 4.3 dargestellt.

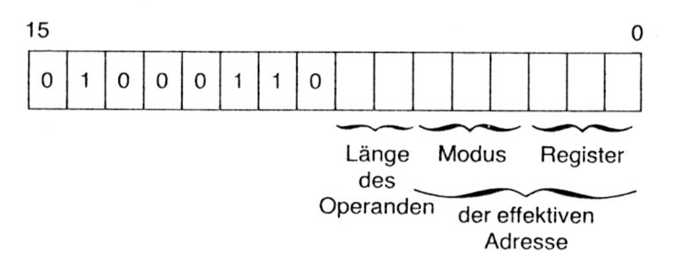


Abb. 4.3: Format des Befehls NOT

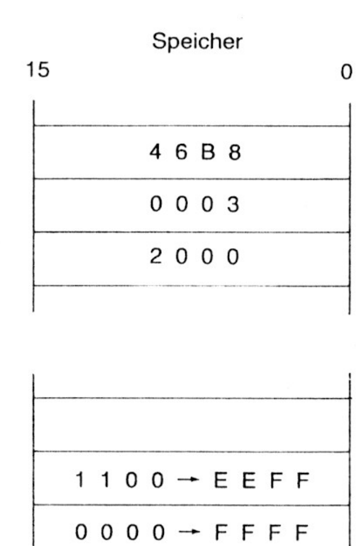
Befehlscode:

0	1	0	0	0	1	1	0	1	0	1	1	1	0	0	1

Hexadezimalcode des Befehls:

46B9
0003
2000

Ausgeführte Aktion:



$32000 und $32001 :

$32002 und $32003 :

DIREKTE REGISTERADRESSIERUNG

Bei dieser Adressierungsart ist der Operand in einem angegebenen Register als effektive Adresse (EA) enthalten.

Datenregister

Dieses Register ist eines der acht 32-Bit-Register Dn. Wir können also schreiben:

EA=Dn

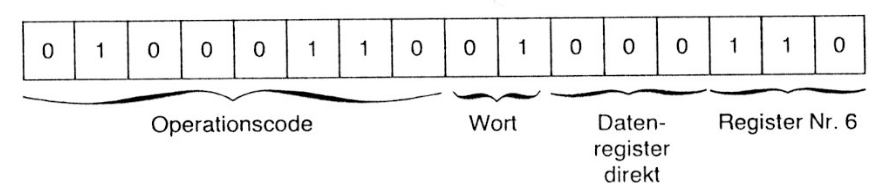
Dn: | Operand |

Beispiel:

NOT D6
(Bilde das Einerkomplement für das
Wort im Register D6)

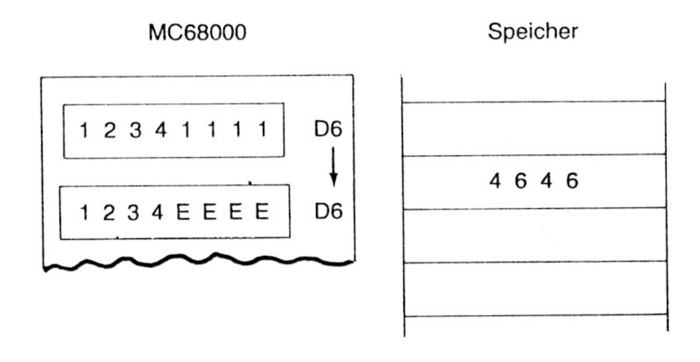
Wenn kein Datenlängencode angegeben ist, wird als Standardwert .W angenommen.

Befehlscode:



Hexadezimalcode:

4646



Für bestimmte Befehle, die zwei effektive Adressen benötigen, z. B. MOVE, ist es möglich, Adressierungsarten zu kombinieren.

Beispiel:

Wir zeigen die gemischte Verwendung der direkten Adressierung des Datenregisters und der absolut kurzen.

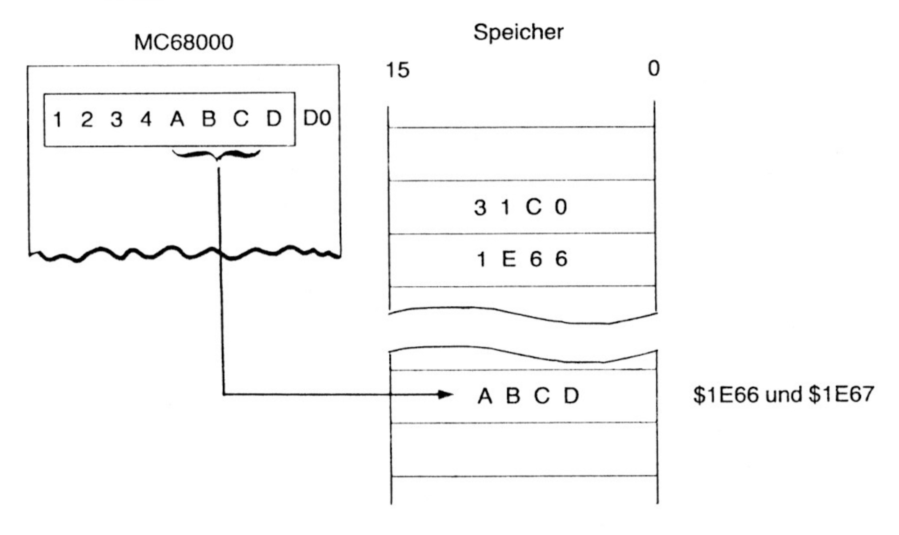
MOVE D0,$1E66
(Übertrage die 16 unteren Bits von
D0 zur Adresse $1E66)

Befehlscode:

0	0	1	1	0	0	0	1	1	1	0	0	0	0	0	0

Hexadezimalcode:

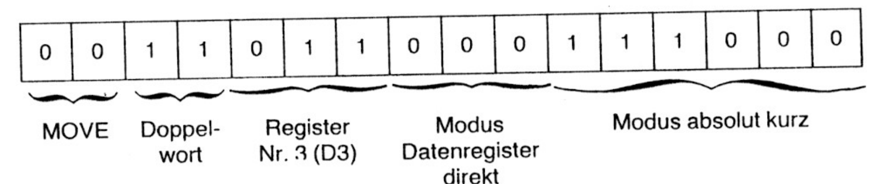
31C0
1E66



Es folgt ein anderes Beispiel, bei dem das Datenregister das Ziel der Übertragung ist.

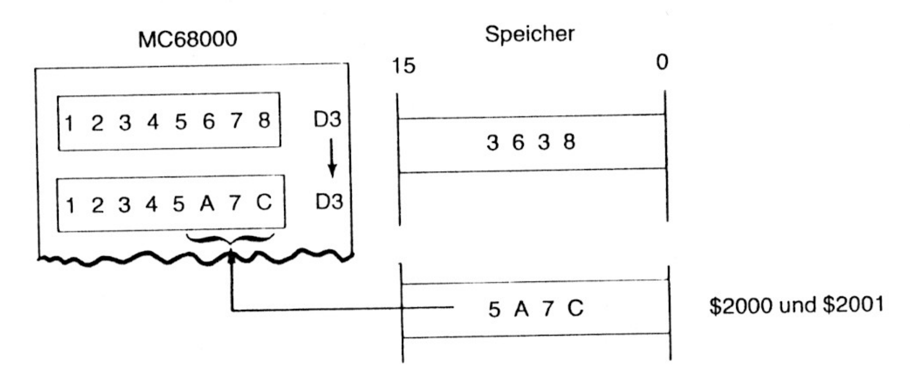
MOVE.W $2000,D3
(Übertrage das an Adresse $2000
stehende Wort ins Datenregister D3)

Befehlscode:



Hexadezimalcode:

3638



Adreßregister

Es gibt sieben 32-Bit-lange Adreßregister An.

EA=An

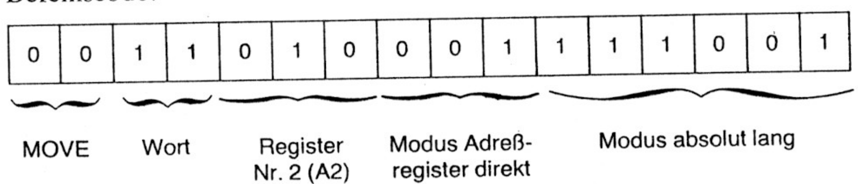
An: Operand

Beispiel:

Das Adreßregister ist der Zieloperand einer Übertragung.

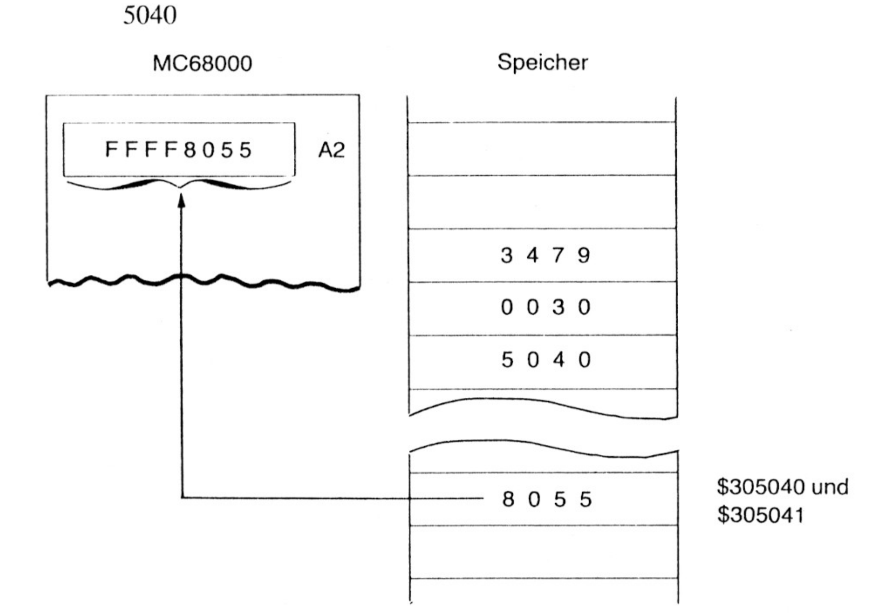
MOVE $305040,A2

Befehlscode:



Hexadezimalcode:

3479
0030
5040



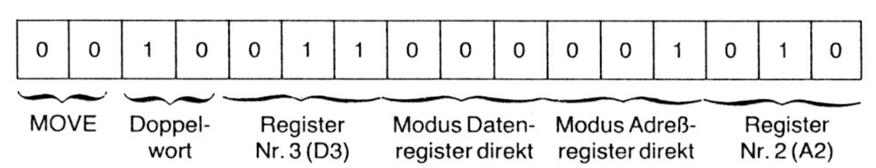
Bitte vergessen Sie nicht, daß bei der Verwendung des Adreßregisters als Zieloperand eine vorzeichenbehaftete Erweiterung erfolgt, wenn es sich um eine Wortoperation handelt.

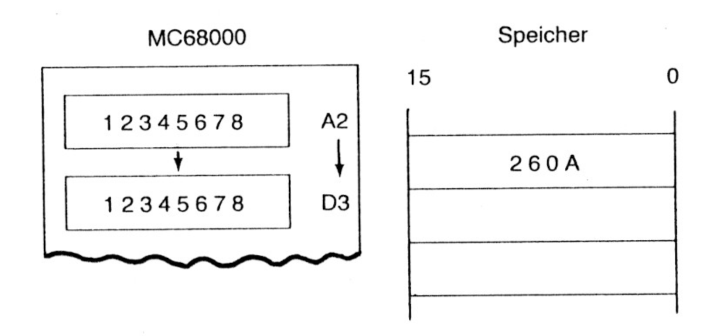
Beispiel:

Das Register An ist im folgenden der Quelloperand.

MOVE.L A2,D3
(Übertrage das Doppelwort aus A2
nach D3)

Befehlscode:





Statusregister

Folgende privilegierte Befehle verwenden die direkte Adressierung:

- ANDI mit SR
- EORI mit SR
- ORI mit SR
- MOVE mit SR

Schreibweise:

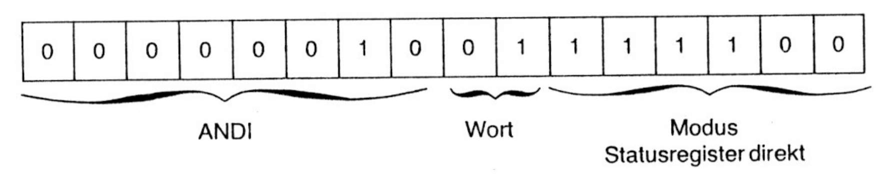
EA=SR

SR: Operand

Beispiel:

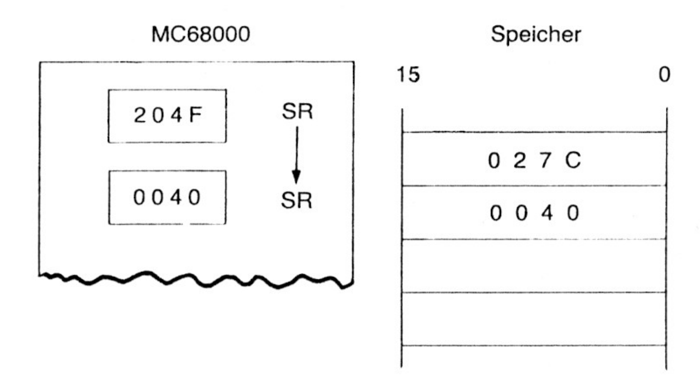
ANDI #$0040, SR

Befehlscode:



Hexadezimalcode:

027C
0040

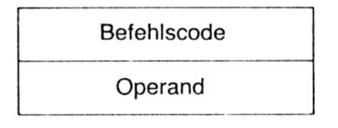


UNMITTELBARE ADRESSIERUNG

Bei dieser Adressierungsart wird der Operand dem Befehl als Konstante mitgegeben.

Einfache unmittelbare Adressierung

Der Operand wird in dem unmittelbar dem Befehlscode folgenden oder den zwei nächsten Worten angegeben.



oder
genauer:

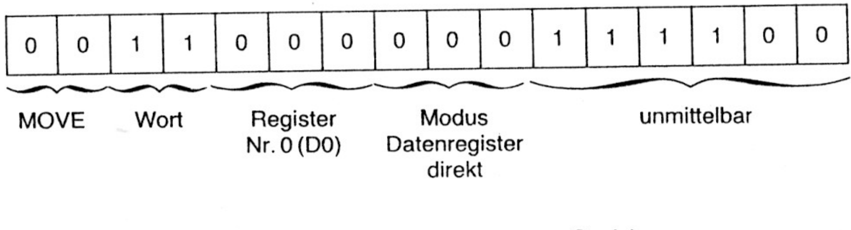
Befehlscode
Höherwertiger Teil des Operanden
Niederwertiger Teil des Operanden

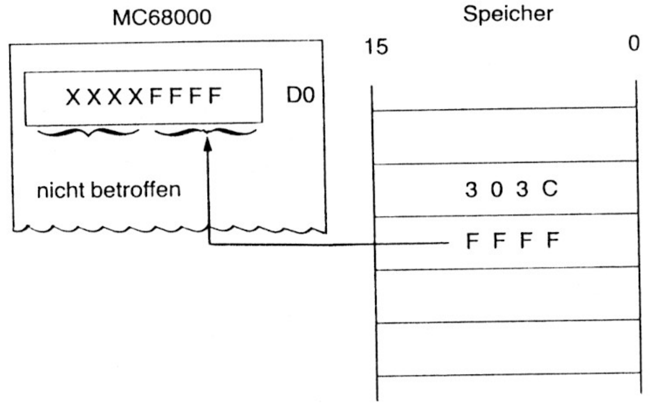
Schreibweise: Unmittelbar xxxx wird als #xxxx angegeben.

Beispiel:

MOVE #$FFFF,D0

Befehlscode:





Wenn der Befehl folgendermaßen gelautet hätte:

MOVE #$FFFF,A0

würde nach der Ausführung das Register A0 den Wert FFFFFFFF wegen der vorzeichenbehafteten Erweiterung enthalten.

Schnelle unmittelbare Adressierung

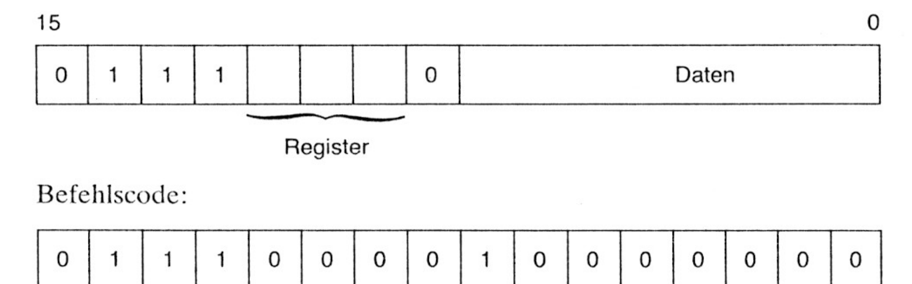
Bei diesem Typ ist die Konstante in den unteren 8 Bits des Befehlswortes selbst enthalten. Daher darf sie nicht länger als 8 Bits sein.

15 8	7 0
Operationscode	8-Bit-Daten

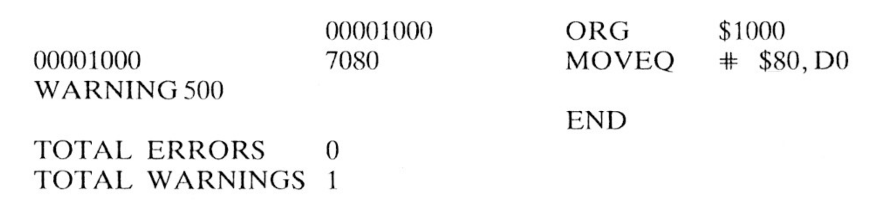
Beispiel:

MOVEQ #$80,D0 (Move Quick)
(Übertrage $80 schnell nach D0 im
Anschluß an die vorzeichenbehaftete
Erweiterung von $80)

Befehlsformat von MOVEQ:



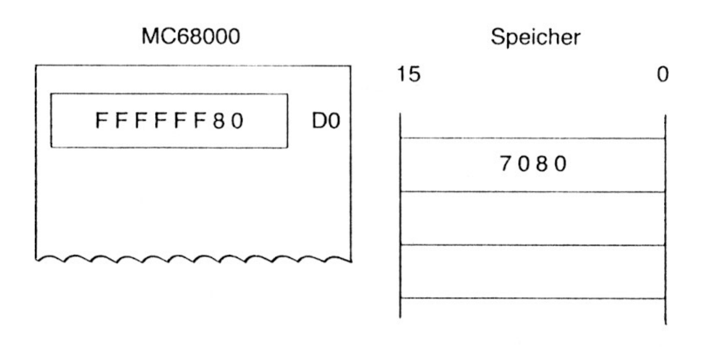
Befehlscode:



Beim Assemblieren des Befehls ergibt sich:

```
                    00001000       ORG      $1000
00001000            7080           MOVEQ    #  $80, D0
WARNING 500
                                   END
TOTAL  ERRORS       0
TOTAL  WARNINGS     1
```

Die Warnung im Assembler-Programm deutet darauf hin, daß eine vorzeichenbehaftete Erweiterung vorgenommen wurde.



Ein anderes Beispiel:

```
                00001000        ORG      $1000
00001000        7070            MOVEQ    # $70, D0
                                END
TOTAL ERRORS    0
TOTAL WARNINGS  0
```

ADRESSIERUNGSART ADRESSREGISTER INDIREKT

Im allgemeinen zeigt bei der indirekten Adressierungsart der Inhalt des Adreßregisters auf einen Operanden.

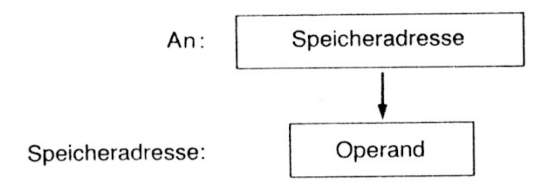
Adreßregister indirekt einfach

Bei dieser Art zu adressieren enthält das Adreßregister die effektive Adresse selbst.

Schreibweise:

EA=(An)

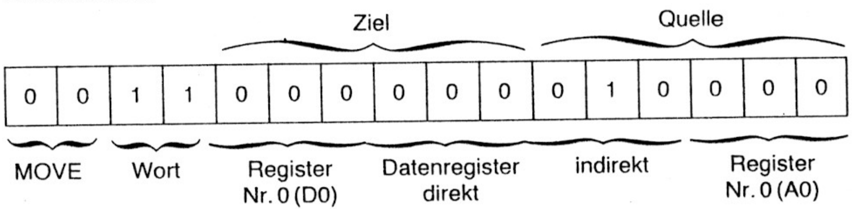
Der Inhalt von An enthält die wirkliche Adresse des Operanden.

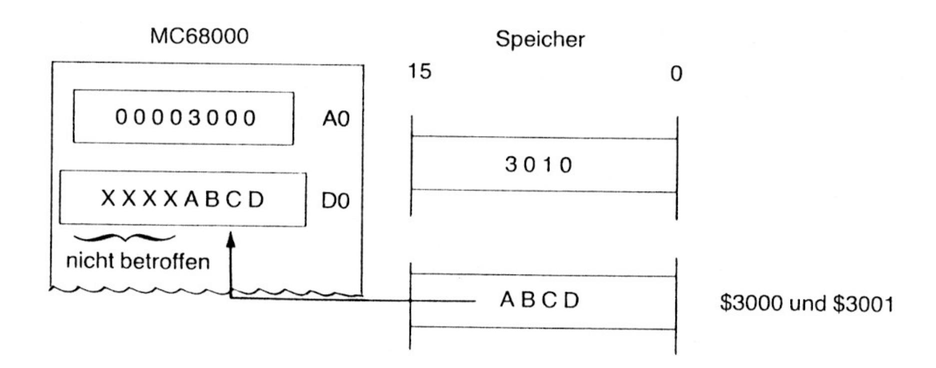


Beispiel:

MOVE (A0),D0
(Übertrage das Wort, das an der mit
A0 adressierten Speicheradresse
steht, nach D0)

Befehlscode:





Adreßregister indirekt mit Postinkrement

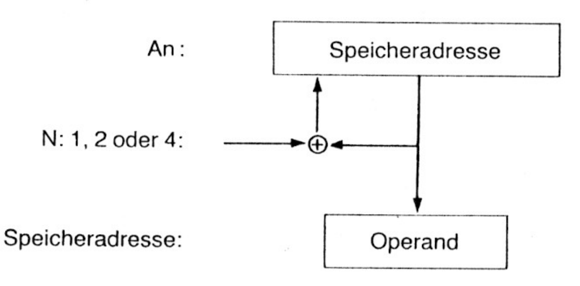
Das im Befehl angesprochene Adreßregister enhält die Adresse des Operanden. Wenn das Datenelement gefunden wurde, wird das Register An um N erhöht.

Dabei kann N folgende Werte annehmen:

N=1, wenn der Operand ein Byte ist
N=2, wenn der Operand ein Wort ist
N=4, wenn der Operand ein Doppelwort ist

Schreibweise:

EA=(An) und An=An+N



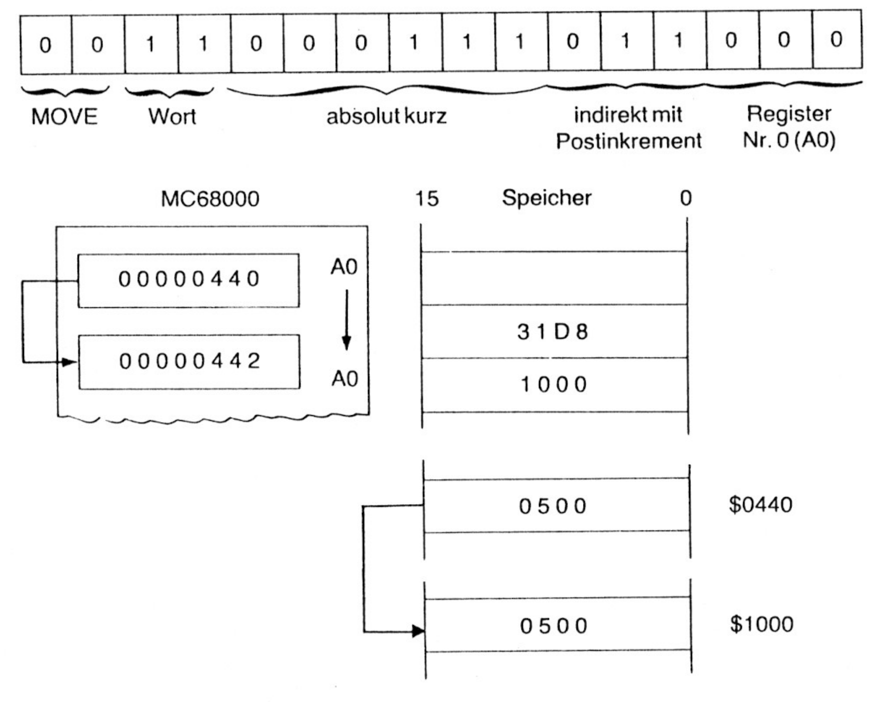
Wir kennzeichnen die Postinkrementierung durch das +-Zeichen.

Beispiele:

```
MOVE.W D0,(A0)+
ADD.L D4,(A1)+
MOVE (A0)+,$1000
```

Dieses letzte Beispiel wollen wir noch etwas näher betrachten.

Befehlscode:



Das Register A0 enthält die Adresse des Operanden: $0440.

Einerseits wird das Wort mit dem Inhalt $0500 an der Adresse $0440 gelesen und eine Kopie davon an die Adresse $1000 geschrieben. Andererseits wird die Adresse $0440 um 2 erhöht, da der MOVE-Befehl eine Wortoperation war. Auf diese Weise enthält A0 am Ende der Übertragung die Adresse $00000442.

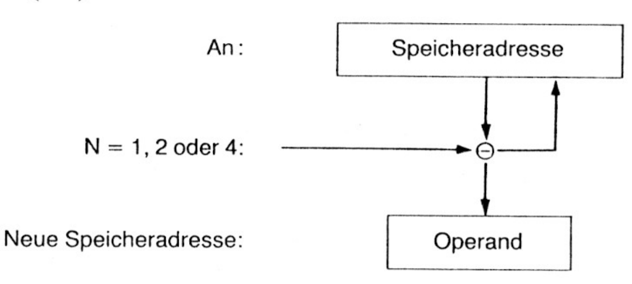
Adreßregister indirekt mit Predekrement

Das Adreßregister An enthält eine Speicheradresse. Diese Adresse muß um N dekrementiert werden, um beispielsweise einen benachbarten Operanden anzusprechen.

N=1, wenn der Operand ein Byte ist
N=2, wenn der Operand ein Wort ist
N=3, wenn der Operand ein Doppelwort ist

Schreibweise:

EA=(An) und An=An−N



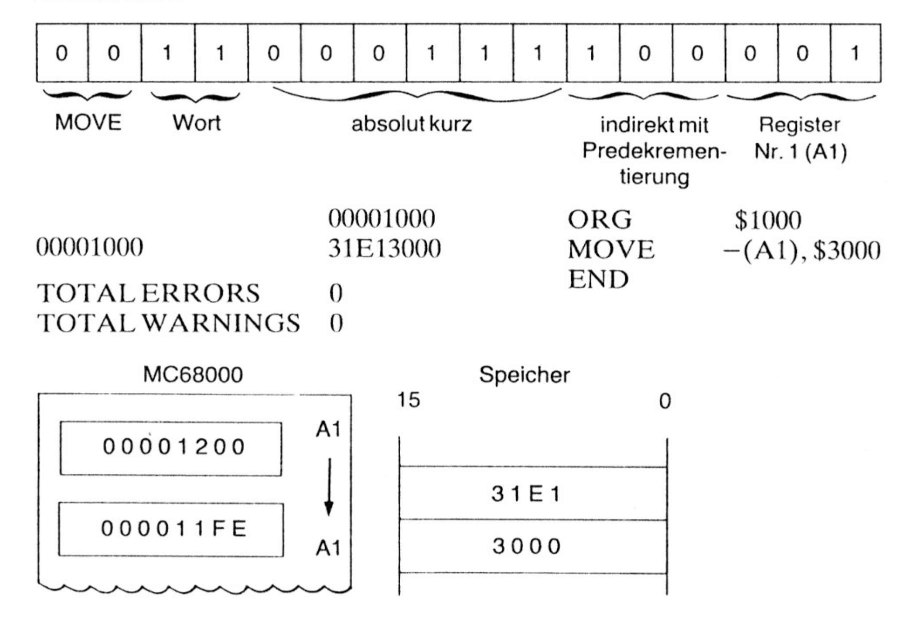
Die Vordekrementierung wird durch ein Minus-Zeichen gekennzeichnet.

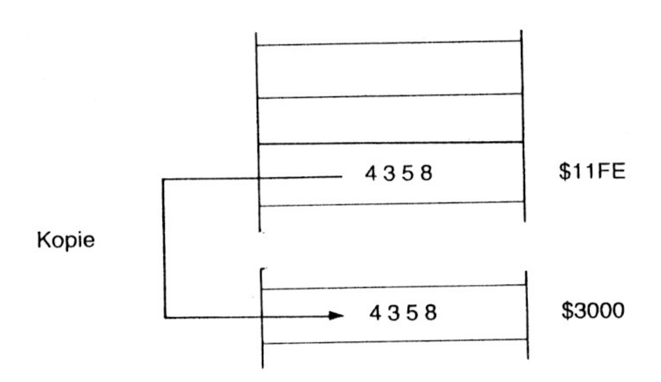
Beispiele:

```
NOT.W −(A0)
AND.L D0,−(A6)
MOVE −(A1),$3000
```

Das letzte Beispiel wollen wir uns wieder genauer ansehen.

Befehlscode:





Das Adreßregister A1 enthält die Speicheradresse $00001200. Diese Adresse wird um 2 vermindert, denn der MOVE-Befehl ist hier eine Wortoperation. A1 ist nun also $000011FE.

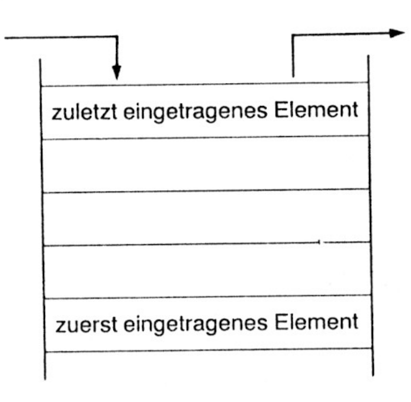
An dieser Adresse finden wir den Operanden $4358, dessen Kopie an die Adresse $3000 übertragen werden soll.

Bedeutung der Nachinkrementierung bzw. Vordekrementierung

Diese beiden Adressierungsarten erlauben dem Prozessor, auf sehr einfache Weise Stapelverarbeitung zu realisieren und Tabellen, Datenfolgen oder Warteschlangen zu bearbeiten.

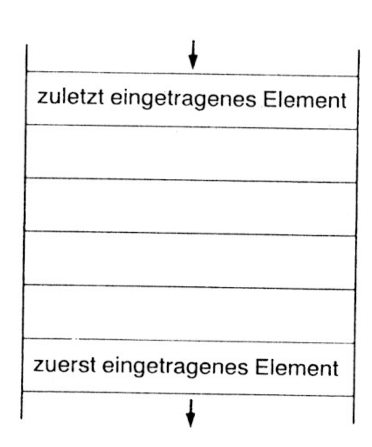
Definition 1:

Ein Stapel (Stack) ist ein Speicher oder Speicherbereich, bei dem die zuletzt abgelegte Information als erstes wieder ausgelesen wird (LIFO: Last In, First Out).



Definition 2:

Eine Warteschlange (Queue) ist ein Speicher oder Speicherbereich, bei dem die zuerst abgelegte Information auch als erstes wieder ausgelesen wird (FIFO: First In, First Out).



Aufbau und Verwendung von Stapeln

Um eine Stapelstruktur zu realisieren, verwendet man hauptsächlich die Adressierungsarten „Adreßregister indirekt mit Postinkrementierung" (An)+ und „Adreßregister indirekt mit Predekrementierung" −(An), wobei die Adreßregister A0, …, A6 die Stapelzeiger darstellen. Dieser Stapel kann von niedrigen zu hohen Adressen gefüllt werden oder umgekehrt.

Der aufsteigende Stapel: Schreiben von hohen zu niedrigen Adressen

Um diesen Stapel zu erzeugen, benötigt man

– die Vordekrementierung zum Füllen des Stapels;

– die Nachinkrementierung zum Leeren des Stapels.

Angenommen unser Stapel soll im Speicherbereich $1000 bis $1009 liegen, und er soll mit Bytes gefüllt werden (Abb. 4.4).

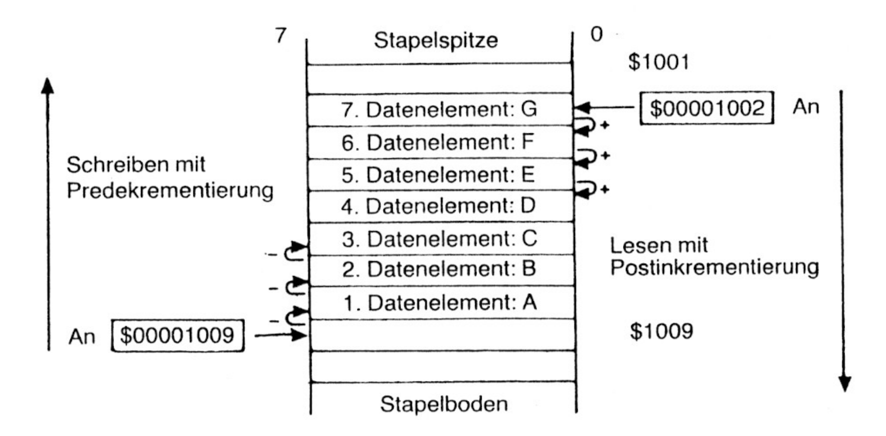


Abb. 4.4: Organisation eines aufsteigenden Stapels

Da der Stapel aufsteigend ist, liegt das zuerst eingetragene Element unten, und wir müssen daher das Register An mit $1009 initialisieren.

Im folgenden wird ein Befehl zur Vordekrementierung gegeben, damit An auf $1008 gesetzt wird, um das Datenelement A dort eintragen zu können.

Die nächstfolgende Predekrementierung bringt An auf $1007 und schreibt das Datenelement B dort hinein.

Auf diese Weise wird ein aufsteigend gefüllter Stapel realisiert.

Bemerkung: Bei diesem Stapelbetrieb ist die letzte Stelle immer leer. Das Adreßregister zeigt immer auf das zuletzt eingetragene Datenelement.

Wenn das siebte Datenelement G eingetragen ist, enthält das Adreßregister also den Wert $1002. Falls wir jetzt den Stapel wieder auslesen wollen, muß notwendigerweise G zuerst herausgenommen werden. Danach wird An erhöht, um das folgende Datenelement lesen zu können. Es handelt sich hierbei um eine Postinkrementierung.

Der absteigende Stapel: Schreiben von niedrigen zu hohen Adressen.

Um diesen Stapel zu erzeugen, füllt man ihn in Richtung der aufsteigenden Adressen.

Wir greifen das vorhergehende Beispiel auf. Im jetzt vorliegenden Fall entspricht die Adresse $1000 dem Stapelboden und $1009 der Stapelspitze (Abb. 4.5).

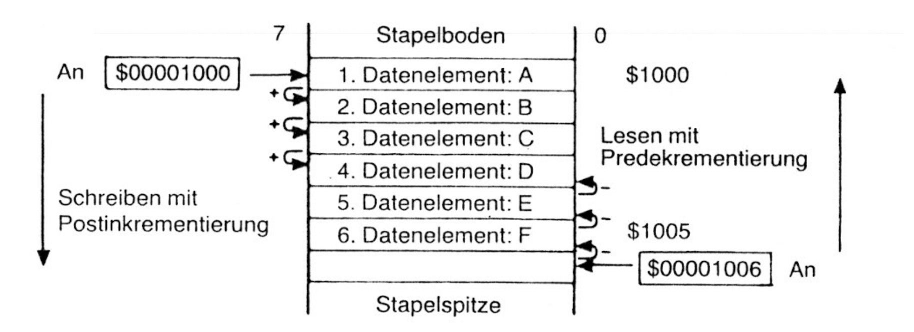


Abb. 4.5: Organisation eines absteigenden Stapels

Damit nun die erste Dateneinheit an die Adresse $1000 geschrieben werden kann, muß An mit $1000 initialisiert werden, worauf dann eine Nachinkrementierung durchgeführt wird. Das Datenelement wird an die Adresse $1000 geschrieben, und danach wird An erhöht. Der Zeiger wird für die Speicherung des nächsten Datenelementes gesetzt.

Bemerkung: An zeigt stets auf das Stapelfeld, das als nächstes beschrieben werden soll. Also wenn man den Stapel jetzt wieder auslesen will, zeigt das Register An noch auf $1006, und das zuletzt geschriebene Datenelement befindet sich an der Adresse $1005. Daher muß An vermindert werden, um an diesen Wert zu gelangen.

Damit der Stapel gelesen werden kann, muß zuvor eine Predekrementierung durchgeführt werden.

Zusammenfassend kann man sagen, daß bei diesem Stapelbetrieb folgendermaßen vorgegangen wird:

- Postinkrementierung, um den Stapel zu füllen.
- Predekrementierung, um den Stapel zu leeren.

Aufbau und Verwendung von Warteschlangen

Die Warteschlangen (Queues) können wie die Stapel in aufsteigender oder absteigender Richtung verwaltet werden. Um solche Datenfolgen zu erzeugen, werden notwendigerweise zwei Adreßregister benutzt; eins zeigt auf das Ende der Warteschlange, wo die neuen Daten eingetragen werden, das andere zeigt auf den Anfang, wo die Daten zuerst entnommen werden.

Die aufsteigende Warteschlange

Dieser Warteschlangentyp wird folgendermaßen realisiert:

- Predekrementierung, um neue Daten in die Warteschlange einzureihen;
- Postinkrementierung, um Daten aus der Warteschlange zu entnehmen.

Die beiden Register An und An' werden jeweils in die gleiche Richtung verändert. Dabei werden sie aber vollkommen unabhängig voneinander verwaltet.

Nach einem Schreibvorgang:

- An zeigt auf die zuletzt eingetragene Dateneinheit in der Warteschlange.
- An' zeigt unverändert auf die letzte Dateneinheit, die die Warteschlange verlassen hat.

Nach einem Lesevorgang:

- An zeigt unverändert auf die letzte Dateneinheit, die in die Warteschlange eingereiht worden ist.

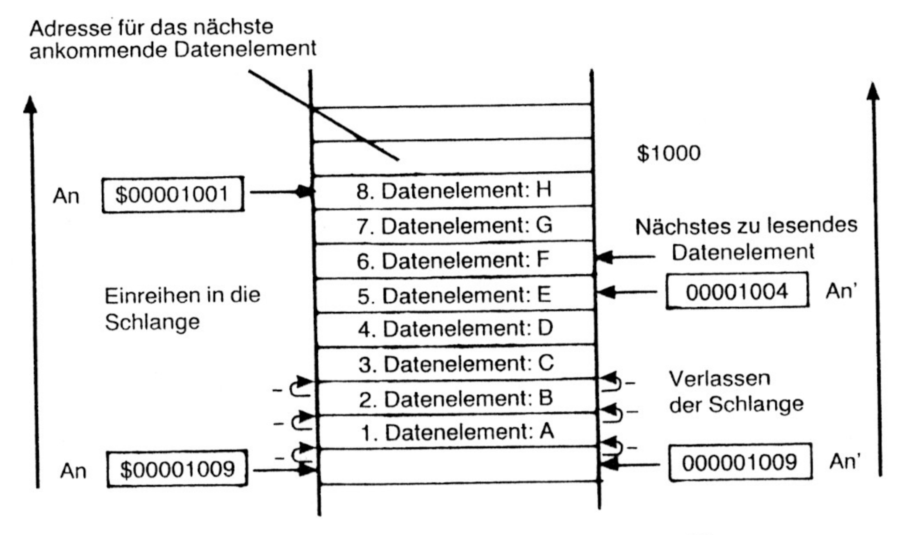


Abb. 4.6: Organisation einer aufsteigenden Warteschlange

– An' zeigt auf die letzte Dateneinheit, die die Warteschlange verlassen hat.

Dieser Vorgang wird noch einmal in der Abb. 4.6 verdeutlicht, die auf den vorangehenden Beispielen basiert.

Die absteigende Warteschlange

Sie wird folgendermaßen realisiert:

– Postinkrementierung, um die Warteschlange zu füllen;

– Postinkrementierung, um die Warteschlange zu leeren.

Auf diese Weise wird das Register An dazu benutzt, um Daten in der Warteschlange abzulegen, und An', um Daten aus ihr zu entnehmen. An und An' werden vollkommen unabhängig voneinander verwaltet.

Nach einem Schreibvorgang:

– An zeigt auf das nächste verfügbare Warteschlangenfeld.

– An' zeigt unverändert auf das Feld, aus dem zuletzt ein Datenelement entnommen wurde.

Nach einem Lesevorgang:

– An zeigt unverändert auf das nächste freie Warteschlangenfeld.

– An' zeigt auf das zuletzt ausgegebene Warteschlangenfeld.

Dieser Vorgang wird durch Abb. 4.7 verdeutlicht.

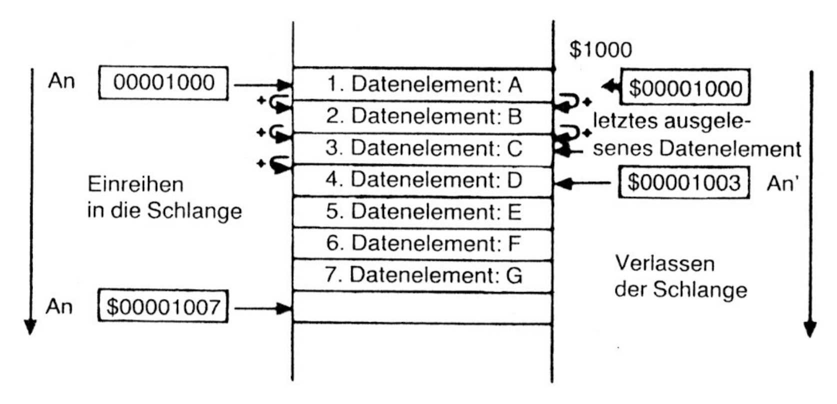


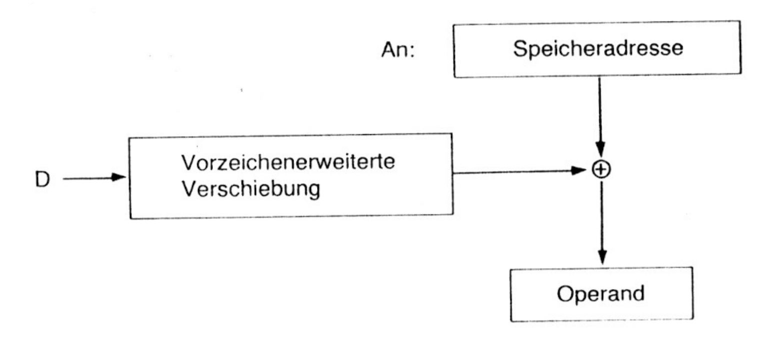
Abb. 4.7: Organisation einer absteigenden Warteschlange

Adreßregister indirekt mit Verschiebung

Die Operandenadresse ergibt sich aus der Summe der Adresse im Adreßregister plus dem 16-Bit-Adreßdistanzwert im vorzeichenbehafteten Erweiterungswort.

Schreibweise:

EA=(An)+D ; D ist der Adreßdistanzwert



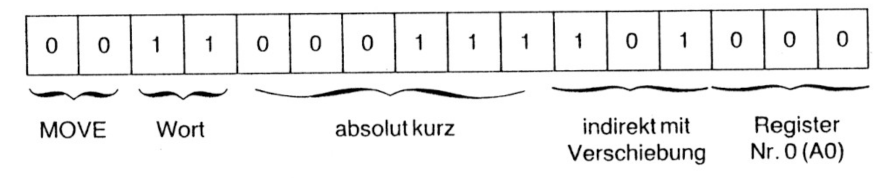
Die Verschiebung wird vor dem Register An angegeben.

Beispiele:

```
ADDI #$FF,8(A0)
MOVE $300(A0),$1000
```

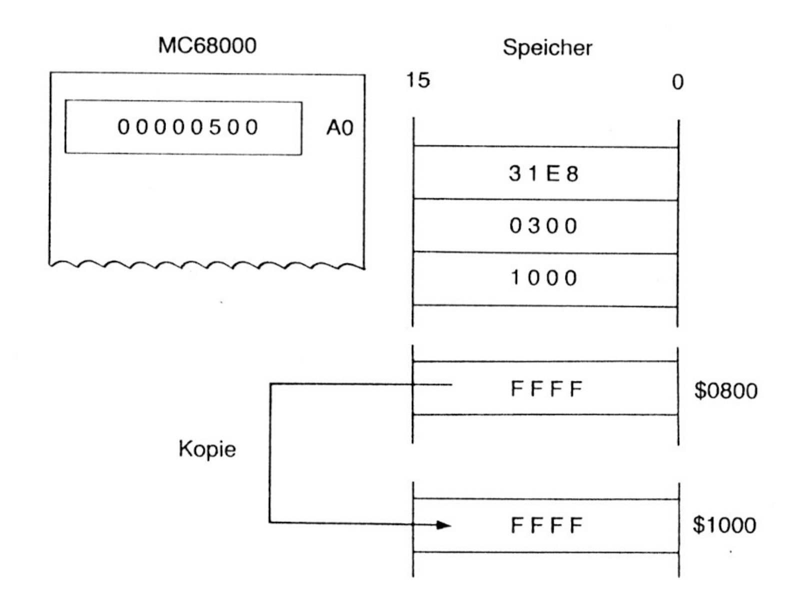
Das letzte Beispiel wollen wir wieder näher erläutern.

Befehlscode:



Hexadezimalcode:

31E8
0300
1000



Berechnen der Adresse:

$$\begin{array}{r} (A0)=00000500 \\ + \\ D=00000300 \\ \hline 00000800 \end{array}$$

Der MOVE-Befehl veranlaßt den Prozessor, die Dateneinheit zu übertragen, wobei in diesem Fall ein Wort von der errechneten Quelladresse \$0800 nach \$1000 kopiert wird. Die Berechnung der Adresse ist durch Addition der Adresse A0 mit Adreßdistanzwert \$300 erfolgt.

Bei der Zieladresse \$1000 handelt es sich um eine absolut kurze Adresse.

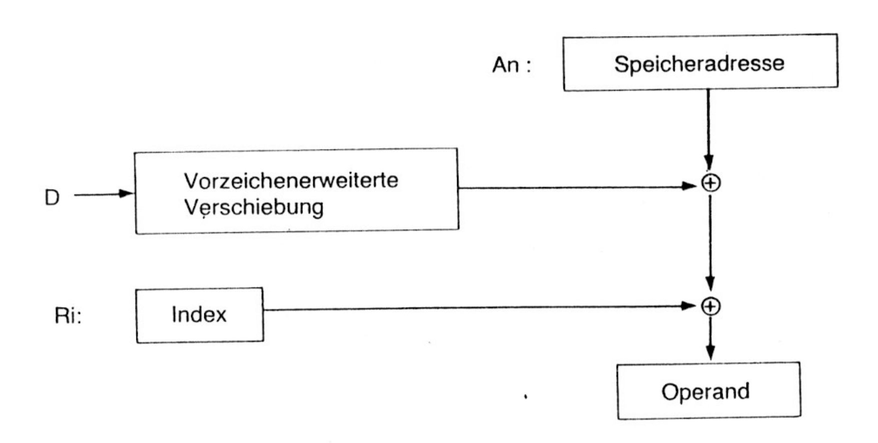
Diese Adressierungsart gestattet es, den Speicher entsprechend einer festen Adresse anzusprechen, um ein Datenelement relativ zu einer Basisadresse aufzufinden.

Adreßregister indirekt mit Index und Adreßdistanzwert

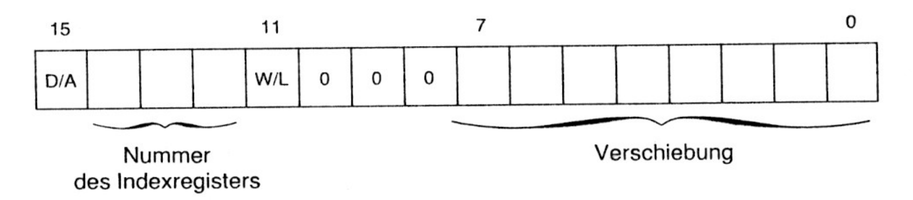
Die Operandenadresse ergibt sich aus der Summe der Adresse im Adreßregister, dem 8-Bit-Adreßdistanzwert aus dem niederwertigen Teil des Erweiterungswortes und dem Inhalt des Indexregisters.

Schreibweise:

EA=(An)+D+(Ri); Ri ist das Indexregister



Diese Adressierungsart erfordert ein Erweiterungswort, dessen Aufbau in der Abb. 4.8 dargestellt wird.



D/A = 0: Das Indexregister ist ein Datenregister.
D/A = 1: Das Indexregister ist ein Adreßregister.

W/L = 0: Der Inhalt des Indexregisters ist ein Wort.
W/L = 1: Der Inhalt des Indexregisters ist ein Doppelwort.

Abb. 4.8: Aufbau eines Erweiterungswortes bei indirekter indizierter Adressierung

Das Bit 11, das W/L genannt wird, gibt die Indexgröße an.

Im Falle eines 16-Bit-Index schreibt man

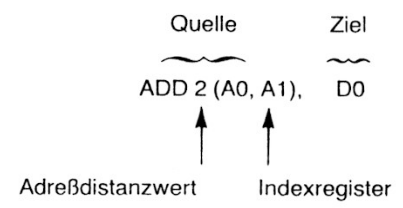
Ri.W oder Ri

Bei der Berechnung der tatsächlichen Adresse wird eine vorzeichenbehaftete Erweiterung des Index auf 32 Bit vorgenommen.

Im Falle eines 32-Bit-Index schreibt man

Ri.L

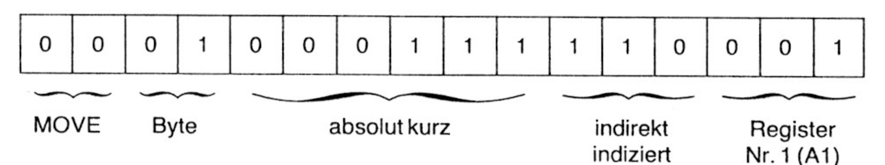
Beispiel:



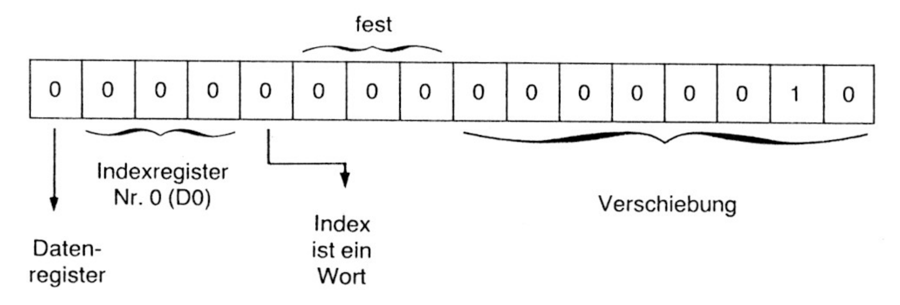
Folgendes Beispiel wollen wir näher betrachten:

MOVE.B 2(A1,D0),$2000

Das erste Wort hat folgende Darstellung:



Das zweite Wort hat folgende Darstellung:



Hexadezimalcode:

11F1
0002
2000

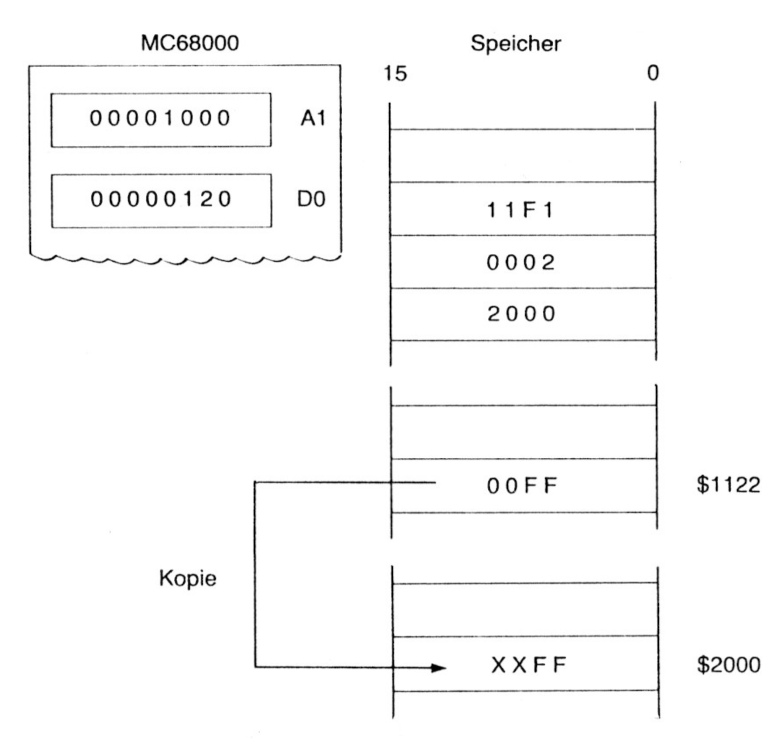
Befehlsbeschreibung: Der zu übertragende Operand ist ein Byte. Der Befehl veranlaßt den Prozessor, die effektive Adresse des Operanden, ausgehend von der Verschiebung 2, dem Adreßregister A1 und dem Indexregister D0 zu berechnen.

Danach wird der Inhalt des so adressierten Speicherfeldes zur absoluten Adresse $2000 gebracht.

Berechnung der effektiven Adresse:

$$EA = (An) + (R:) + D$$
$$EA = (A1) + (D0) \quad 2$$

$$\begin{array}{r} (A1) = 00001000 \\ + \\ (D0) = 00000120 \\ + \\ D = 00000002 \\ \hline 00001122 \end{array}$$



Vorteile dieser Adressierungsart: Oft ist es notwendig, auf den Inhalt einer Tabelle im Speicher zugreifen zu können. Man wählt dann ein Adreßregister, das z. B. die Basisadresse dieser Tabelle enthält. Die einfache indizierte Adressierung (wobei der Adreßdistanzwert D=0 ist) ermöglicht dann, zur Basisadresse einen Index zu addieren, so daß jeweils auf das n-te Element der Tabelle (Byte, Wort oder Doppelwort) zugegriffen werden kann. Mit Hilfe des Adreßdistanzwertes ist es dann möglich, relativ zu einer Tabelle weitere zu erzeugen.

ADRESSIERUNGSARTEN RELATIV ZUM PROGRAMMZÄHLER

Bei dieser Adressierungsart ist die effektive Adresse eine Funktion des Wertes des Programmzählers und eines 16-Bit-Distanzwertes.

Es gibt zwei Arten von relativer Adressierung: relative Adressierung, bezogen auf den Programmzähler, und relativ indizierte Adressierung, bezogen auf den Programmzähler.

Der Sinn der relativen Adressierung liegt darin, eine einfache Verschiebbarkeit und Positionsunabhängigkeit zu erreichen. Außerdem wird durch Vereinfachung der Referenzbezüge zu Nachbaradressen des aktuell bearbeiteten Befehls die erforderliche Befehlslänge deutlich verringert.

Relative Adressierung, bezogen auf den Programmzähler, mit Adreßdistanz

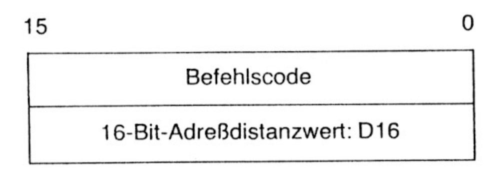
Alle Befehle, die die Programmzähler-relative Adressierung verwenden, benötigen ein Erweiterungswort, um den 16-Bit-Adreßdistanzwert aufzunehmen. Eine Ausnahme bilden dabei die Verzweigungsbefehle, bei denen ein 8-Bit-Adreßdistanzwert einen Teil des Befehlswortes selbst ausmacht.

Dieses Distanzwort wird zum Zeitpunkt der Assemblierung errechnet, wenn zuvor die relative Startadresse des Programmteils mit Hilfe einer speziellen Assembler-Anweisung festgelegt wurde (siehe auch RORG).

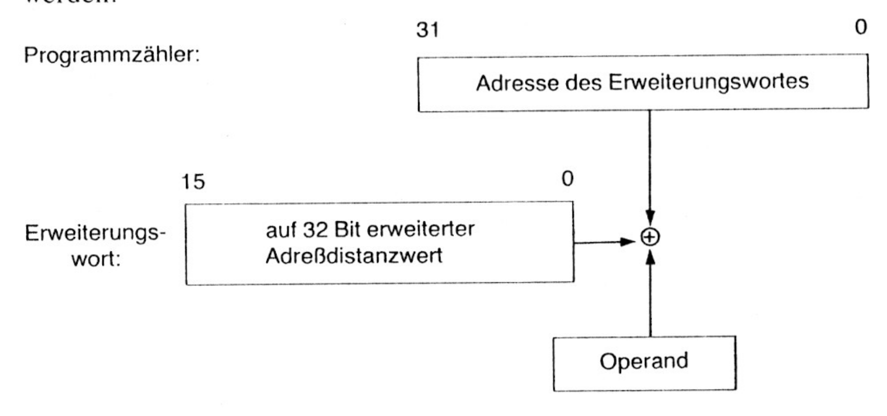
Schreibweise:

EA=(PC)+D16

Format von Befehlen mit relativer Adressierung:



Zum Zeitpunkt der Ausführung eines solchen Befehls zeigt der Befehlszähler schon auf das folgende Wort, das den Distanzwert enthält. Der Zugriff auf den Distanzwert kann mit dem folgenden Schema dargestellt werden.



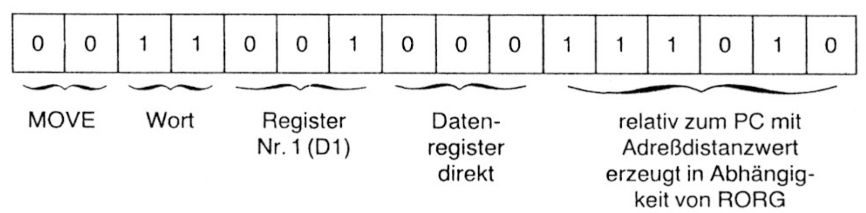
Beispiele für relative Adressierung mit Distanzwert:

Folgendes Programm wird zugrundegelegt:

```
                RORG    $1000
$1000           MOVE    VERSUCH, D1
$1004           NOP                     ; NO oPeration
$1006           NOP
$1008           NOP
$100A VERSUCH   DC      $3030           ; Konstante
                                        ; definieren,
                                        ; Assembler-
                                        ; direktive

                END
```


Befehlscode von MOVE VERSUCH,D1:



Berechnung des Distanzwertes: Die Adresse des Distanzwertes ist $1002. Es muß die Adresse $100A des LabelsVERSUCH angesprungen werden. Der Adreßdistanzwert muß daher $0008 betragen. Das Programm sieht jetzt mit Hex-Codes folgendermaßen aus:

```
                                    RORG    $1000
$1000    323A0008                   MOVE    VERSUCH,D1
$1004    4E71                       NOP
$1006    4E71                       NOP
$1008    4E71                       NOP
$100A    3030         VERSUCH       DC      $3030
                                    END
```

Auf diese Weise ist das Label relativ zur Adresse $1002 festgelegt worden, die die aktuelle Adresse im Programmzähler zum Zeitpunkt der Ausführung des MOVE-Befehls darstellt. Schließlich bewirkt der MOVE-Befehl, daß der Wert $3030 nach D1 geladen wird. Wir stellen fest, daß das Label VERSUCH nach dem MOVE-Befehl festgelegt wird, was die Errechnung des Adreßdistanzwertes vereinfacht.

Wir greifen nochmals das gleiche Beispiel auf. Diesmal jedoch wollen wir die Adresse des Labels VERSUCH vor dem MOVE-Befehl festlegen.

```
                         RORG     $1000
$1000    VERSUCH         DC       $3030
$1002                    NOP
$1004                    NOP
$1006                    NOP
$1008                    MOVE     VERSUCH,D1
                         END
```

Was den MOVE-Befehl betrifft, ist der Operationscode und die Angabe der effektiven Adresse die gleiche geblieben. Allerdings hat sich der Betrag des Distanzwertes verändert.

Der Distanzwert muß nun von der Adresse $100A (die aktuelle Adresse zum Zeitpunkt der Ausführung des MOVE-Befehls) abgezogen werden, damit das Label an der Adresse $1000 aufgefunden werden kann.

Der Adreßabstand zwischen dem MOVE-Befehl und dem Label beträgt $000A; es genügt also, das Zweierkomplement dieses Wertes zu bilden, um den neuen Adreßdistanzwert zu erhalten.

Testrechnung:

Adresse im PC	$0000100A
	+
Adreßdistanz	$FFFFFFF6
	$00001000

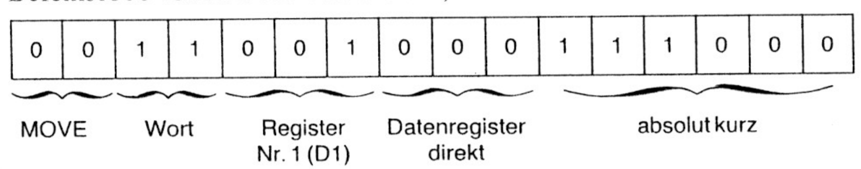
Das zugehörige Programm sieht also folgendermaßen aus:

```
                                   RORG      $1000
$1000    3030       VERSUCH        DC        $3030
$1002    4E71                      NOP
$1004    4E71                      NOP
$1006    4E71                      NOP
$1008    323AFFF6                  MOVE      VERSUCH, D1
                                   END
```

Um die Rolle des Assembler-Befehls RORG deutlich zu machen, lassen wir das erste Beispiel noch einmal mit dem Befehl ORG zu Beginn assemblieren.

```
                     ORG      $1000
$1000                MOVE     VERSUCH, D1
$1004                NOP
$1006                NOP
$1008                NOP
$100A    VERSUCH     DC       $3030
                     END
```

Befehlscode von MOVE VERSUCH,D1:



Hexadezimalcode:

3238
100A (Adresse von VERSUCH)

Das Programm sieht dann so aus:

			ORG	$1000
$1000	3238100A		MOVE	VERSUCH, D1
$1004	4E71		NOP	
$1006	4E71		NOP	
$1008	4E71		NOP	
$100A	3030	VERSUCH	DC	$3030
			END	

Spezialfall: Verzweigungsbefehle

Bei den bedingten Verzweigungen wird, wenn die festgelegte Bedingung eintritt, die Ausführung des Programms an der Adresse (PC)+Adreßdistanzwert fortgesetzt. Andernfalls läuft das Programm fortlaufend ab.

Für alle Verzweigungsarten kann der Adreßdistanzwert in 8 Bit verschlüsselt werden.

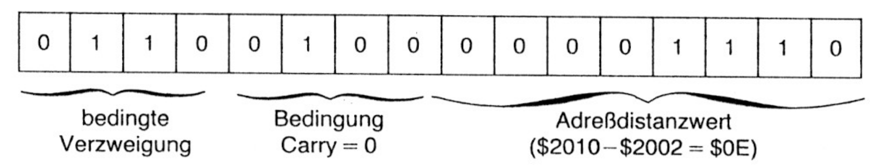
Wenn die Adreßdistanz im Befehlswort den Wert 0 hat, bedeutet das, daß der Adreßdistanzwert im Erweiterungswort in 16 Bits verschlüsselt ist.

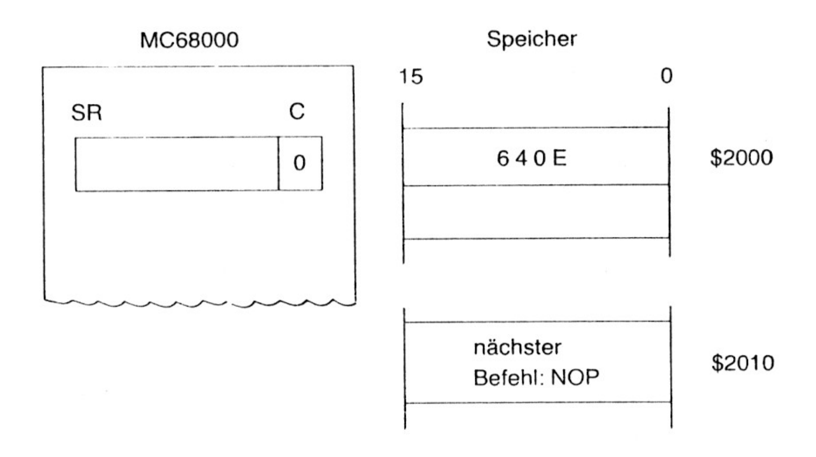
Beispiel 1:

```
$2000               BCC     PROG1
  .                  .
  .                  .
  .                  .
$2010     PROG1     NOP
```

Bedeutung des Befehls BCC: Wenn das Carry-Bit 0 ist, verzweigt der Prozessor zur Adresse PROG1, andernfalls setzt er die Verarbeitung mit dem nächsten Befehl fort.

Befehlscode:



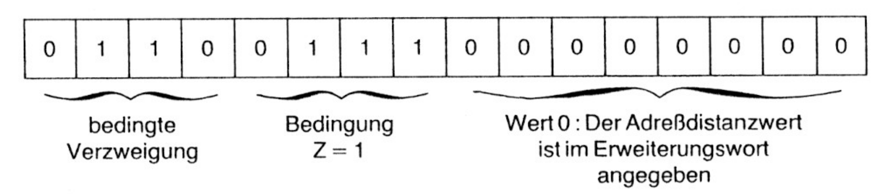


Beispiel 2:

$1000		BEQ	PROG1
.		.	
.		.	
.		.	
.		.	
$2010	PROG1	NOP	

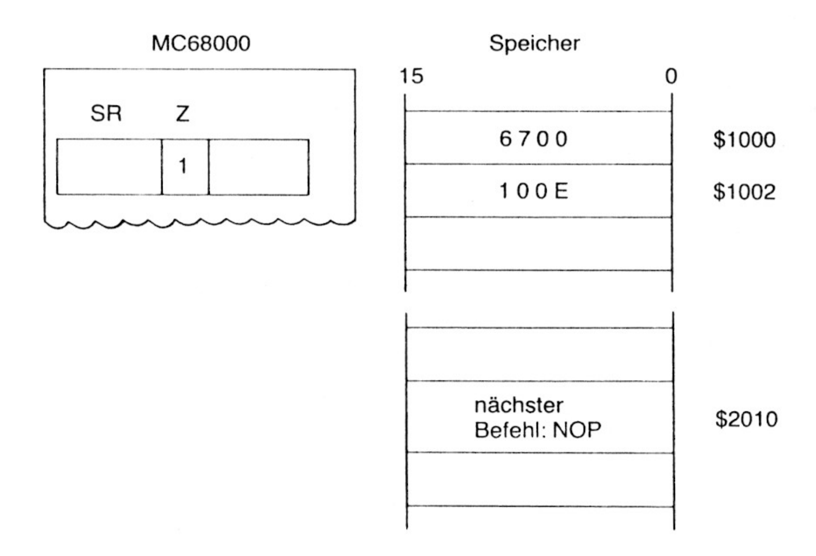
Bedeutung des Befehls BEQ: Es wird zur Adresse $2010 verzweigt, wenn die Überprüfung auf Gleichheit erfolgreich war, d. h. wenn der Bedingungscode Z=1 ist.

Befehlscode:



Code des Erweiterungswortes:

Adresse der Verzweigung:	$2010	
Wert von PC:	–$1002	
Adreßdistanzwert:	$100E	16 Bits



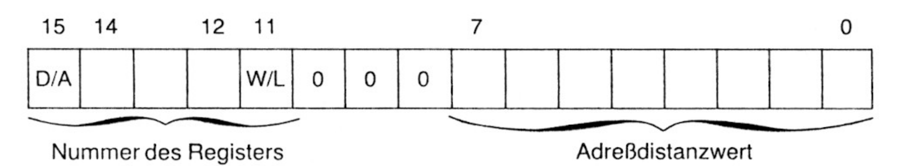
Relative Adressierung mit Index, bezogen auf den Programmzähler

Die Operandenadresse ist die Summe des Wertes, der im Programmzähler enthalten ist, dem Wert aus dem Indexregister und einem 8-Bit-Adreßdistanzwert.

Schreibweise:

$$EA=(PC)+(Ri)+D8$$

Diese Adressierungsart benötigt ein Erweiterungswort, dessen Aufbau in Abb. 4.9 zu sehen ist.



bit 15:

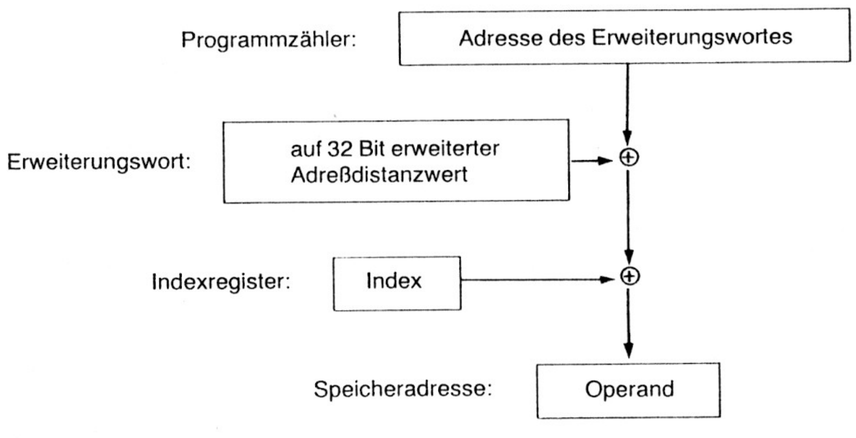
D/A = 0: Das Indexregister ist ein Datenregister.
D/A = 1: Das Indexregister ist ein Adreßregister.

bit 11:

W/L = 0: Das Indexregister enthält ein Wort, das auf 32 Bit vorzeichenerweitert werden muß.
W/L = 1: Das Indexregister enthält ein Doppelwort.

Abb. 4.9: Format des Erweiterungswortes bei der relativen Adressierung mit Index

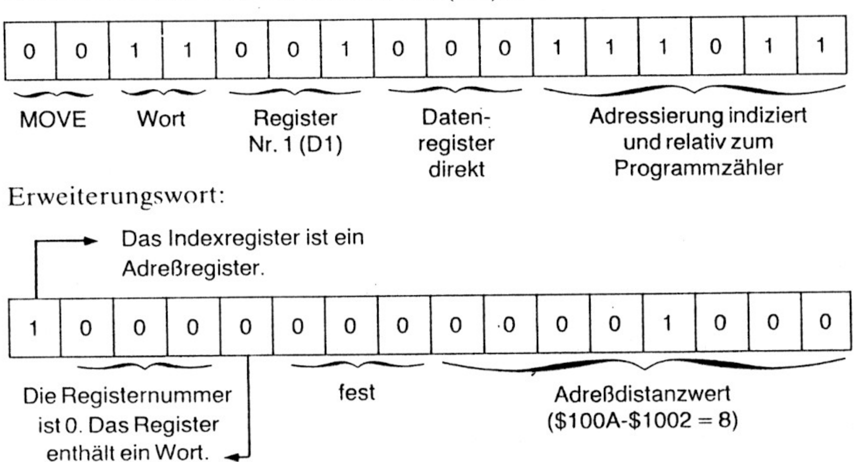
Prinzipielle Arbeitsweise dieser Adressierungsart:



Beispiel:

```
                RORG    $1000
$1000           MOVE    VERSUCH (A0), D1
$1004           NOP
$1006           NOP
$1008           NOP
$100A VERSUCH   DC      $3030
                END
```

Befehlscode von MOVE VERSUCH (A0),D1:



Hexadezimalcode:

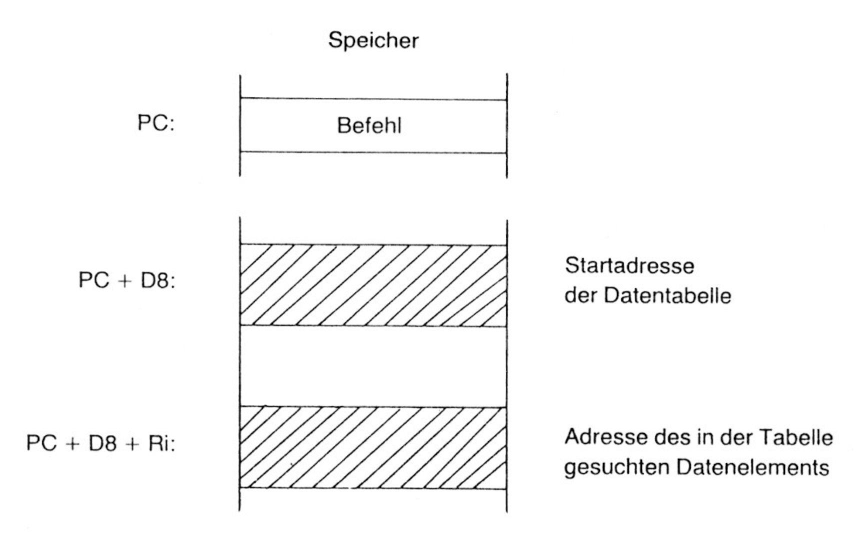
323B
8008

Das zugehörige Programm sieht folgendermaßen aus:

```
                  RORG    $1000
$1000             MOVE    VERSUCH(A0),D1
$1004             NOP
$1006             NOP
$1008             NOP
$100A VERSUCH     DC      $3030
                  END
```

Wenn das Register A0 den Wert 0 enthält, wird das Datenelement $3030 in das Register D1 geschrieben. Wenn A0 von 0 verschieden ist, wird ein anderer Wert in D1 geschrieben.

Bei diesem Adressierungstyp muß man auch beachten, daß das Label (VERSUCH in unserem Fall) als Anfangsadresse einer Datenfolge betrachtet wird, in der man Informationen mit Hilfe des Indexregisters A0 auffindet.



Eine zusammenfassende Tabelle der verschiedenen Adressierungsarten ist in Abb. 4.10 zu sehen.

Adressierungsart	Syntax	Effektive Adresse (EA)
Absolut kurz	16-Bit-Adresse	EA = 16-Bit-Adresse
Absolut lang	24-Bit-Adresse	
Datenregister direkt	DN	EA = DN
Adreßregister direkt	AN	EA = AN
Statusregister direkt	SR	EA = SR
Unmittelbar	#Datenwert 16 oder 32 Bit	Keine Errechnung der effektiven Adresse. Operand ist der Datenwert.
Unmittelbar schnell	#8-Bit-Datenwert	Keine Errechnung der effektiven Adresse. Operand ist der Datenwert.
Indirekt	(AN)	EA = (AN)
Indirekt mit Postinkrementierung	(AN)+	1. EA = (AN) 2. AN = AN + n (n = 1, 2, 4)
Indirekt mit Predekrementierung	−(AN)	1. AN = AN − n (n = 1, 2, 4) 2. EA = (AN)
Indirekt mit Verschiebung	D16 (AN)	EA = (AN) + auf 32 Bit vorzeichenerweiterter 16-Bit-Adreßdistanzwert
Indirekt indiziert mit Verschiebung	D8 (AN,R:)	EA = (AN) + (R:) + auf 32 Bit vorzeichenerweiterter 8-Bit-Adreßdistanzwert
Relativ (zum PC)	(PC) abhängig von Assembler-Anweisung	EA = (PC) + auf 32 Bit vorzeichenerweiterter 16-Bit-Adreßdistanzwert
Relativ (zum PC) indiziert	(PC) (R:) abhängig von Assembler-Anweisung	EA = (PC) + (R:) + auf 32 Bit vorzeichenerweiterter 8-Bit-Adreßdistanzwert

Abb. 4.10: Die Adressierungsarten des MC68000

Übungen

4.1: Welche Länge darf der Operand eines Adreßregisters haben?

4.2: Angenommen wir haben es mit einer Wortoperation zu tun. Welche Rolle spielt das Register An, wenn es die Quelle der Operation bzw. wenn es das Ziel ist?

4.3: Welche Länge hat der Adreßdistanzwert bei der indirekten Adressierung mit Adreßdistanz?

4.4: Welche Länge hat der Index bei der indizierten Adressierung?

4.5: Welche Länge hat der Adreßdistanzwert bei der indirekten Adressierung mit Index?

4.6: Wann verwendet man die Adressierungsarten mit Postinkrementierung bzw. mit Predekrementierung?

4.7: Welchen Sinn hat die indizierte Adressierung?

4.8: Ist die FIFO-Struktur bezeichnend für den Stapel- oder den Warteschlangenbetrieb?

4.9: Wie lang ist der Adreßdistanzwert bei der relativen Adressierung mit Adreßdistanz?

4.10: Um wieviel wird bei der indirekten Adressierung mit Postinkrementierung erhöht, wenn es sich um eine Doppelwort-Operation handelt?

Lösungen

4.1: Da bei Adreßregistern der Datenlängencode .B verboten ist, dürfen die Operanden nur Worte oder Doppelworte sein.

4.2: An als Quelle: Die 16 niederwertigen Bits werden genommen.
An als Ziel: Das gesamte Register wird verändert.

4.3: Der Adreßdistanzwert ist 16 Bits lang.

4.4: Bei der indizierten Adressierung kann der Index ein Wort oder ein Doppelwort sein.

4.5: Bei der indirekten Adressierung mit Index beträgt die Länge des Adreßdistanzwertes 8 Bits.

4.6: Adressierung mit Postinkrementierung und Predekrementierung verwendet man bei der Stapel- und Warteschlangenverarbeitung.

4.7: Indizierte Adressierung erlaubt den einfachen Zugriff zu Datenelementen in Tabellen.

4.8: FIFO nennt man die Struktur bei einem Warteschlangenbetrieb, beim Stapelbetrieb war es die LIFO-Struktur.

4.9: Bei der relativen Adressierung beträgt die Länge des Adreßdistanzwertes 16 Bits, aber speziell bei Verzweigungsbefehlen nur 8 Bits.

4.10: Die Erhöhung beträgt 4.

Kapitel 5

Der Befehlssatz des 68000

Der Befehlssatz des Mikroprozessors 68000 umfaßt 56 Grundbefehle.

Jeder Befehl, abgesehen von einigen Ausnahmen, operiert mit Bytes, Worten oder Doppelworten. Außerdem kann der größte Teil der Befehle alle 14 Adressierungsarten benutzen.

Auf diese Weise – mit den verschiedenen Befehlstypen, den unterschiedlichen Daten und den Adressierungsarten – sind etwa 1000 unterschiedliche Befehlskombinationen möglich.

Dennoch braucht sich der Benutzer nur die mnemonische Schreibweise (sinnvolle Abkürzung) des Befehls zu merken, die Adressierungsart der Quelle und des Ziels je nach der Operation festzulegen und die Länge der Operanden anzugeben.

Was die interne Struktur des Prozessors betrifft, so ist sie sehr anpassungsfähig, da die Adreßregister mit verschiedenen Adressierungsarten arbeiten können (direkt, indirekt, indiziert etc.) und die Indexregister Daten- oder Adreßregister sein können. Andererseits ist jedes Register vollkommen unabhängig von den anderen.

Dank all dieser Eigenschaften ist der Befehlssatz leistungsstark, anpassungsfähig und ermöglicht somit eine effiziente Programmierung des MC68000.

Die Befehle lassen sich folgendermaßen in sieben Kategorien einteilen:

- Befehle für ganzzahlige Arithmetik
- Befehle für binärcodierte Dezimalarithmetik (BCD)
- logische Befehle
- Schiebe- und Rotierbefehle
- Bitmanipulationsbefehle
- Datentransportbefehle
- Systemsteuerbefehle

Logische Befehle

Befehl	Länge des Operanden	Operation
AND	8, 16, 32	DN UND (EA) → DN
		(EA) UND DN → EA
		(EA) UND #Dateneinheit → EA
OR	8, 16, 32	DN ODER (EA) → DN
		(EA) ODER DN → EA
		(EA) ODER #Dateneinheit → EA
EOR	8, 16, 32	(EA) EXKLUSIV-ODER Dy → EA
		(EA) EXKLUSIV-ODER #Dateneinheit → EA
NOT	8, 16, 32	Einerkomplement von (EA) → EA

Arithmetische Befehle

Befehl	Länge des Operanden	Operation
ADD	8, 16, 32	DN + (EA) → DN
		(EA) + DN → EA
		(EA) + #Dateneinheit → EA
		AN + (EA) → AN
ADX	8, 16, 32	Dx + Dy + X → Dx
	16, 32	(−(Ax)) + (−(Ay)) + X → −(Ax)
CLR	8, 16, 32	0 → EA
CMP	8, 16, 32	DN − (EA)
		(EA) − #Dateneinheit
		((Ax)+) − ((Ay)+)
	16, 32	AN − (EA)
DIVS	32 / 16	DN / (EA) → DN
DIVU	32 / 16	DN / (EA) → DN
EXT	8 16	Erweiterung von Bit 7 auf Bits 8 bis 15
	16 32	Erweiterung von Bit 15 auf Bits 16 bis 31
MULS	16 * 16 32	DN * (EA) → DN
MULU	16 * 16 32	DN * (EA) → DN
NEG	8, 16, 32	0 − (EA) − EA
NEGX	8, 16, 32	0 − (EA) − X → EA
SUB	8, 16, 32	DN − (EA) → DN
		(EA) − DN → EA
		(EA) − #Dateneinheit → EA
	16, 32	AN − (EA) → AN
SUBX	8, 16, 32	Dx − Dy − X → Dx
		(−(Ax)) − (−(Ay)) − X → − (Ax)
TAS	8	(EA) − 0 und Bit 7 von EA = 1
TST	8, 16, 32	(EA) − 0

Abb. 5.1: Der Befehlssatz, gegliedert nach Typen

Bevor wir nun einige Befehle und Programmbeispiele detailliert betrachten, stellen wir eine vollständige Befehlsliste zusammen.

Diese Befehlsliste ist auch wieder in die schon erwähnten Kategorien eingeteilt, da dies bei der Erstellung eines Programms eine große Rolle spielt (Abb. 5.1).

Schiebe- und Rotierbefehle

Befehl	Länge des Operanden	Operation
ASL	8, 16, 32	X/C ← ← 0
ASR	8, 16, 32	→ X/C
LSL	8, 16, 32	X/C ← ← 0
LSR	8, 16, 32	0 → → X/C
ROL	8, 16, 32	C ← ←
ROR	8, 16, 32	→ C
ROXL	8, 16, 32	C ← ← X
ROXR	8, 16, 32	X → → C

Bittestbefehle

Befehl	Länge des Operanden	Operation
BTST	8, 32	Test eines Bits von (EA) → Z
BSET	8, 32	Test eines Bits von (EA) → Z und Setzen des getesteten Bits auf 1.
BCLR	8, 32	Test eines Bits von (EA) → Z und Setzen des getesteten Bits auf 0.
BCHG	8, 32	Test eines Bits von (EA) → Z und Invertieren des getesteten Bits.

Abb. 5.1: Der Befehlssatz, gegliedert nach Typen (Forts.)

Befehle für BCD-Arithmetik

Befehl	Länge des Operanden	Operation
ABCD	8	$(Dx)_{10} + (Dy)_{10} + X \rightarrow Dx$ $-(Ax)_{10} + -(Ay)_{10} + X \rightarrow -(Ax)$
SBCD	8	$(Dx)_{10} - (Dy)_{10} - X \rightarrow Dx$ $(Ax)_{10} - -(Ay)_{10} - X \rightarrow -(Ax)$
NBCD	8	$0 - (EA)_{10} - X \rightarrow EA$

Befehle zur Kontrolle des Programmflusses

Befehl	Operation
Bcc	Bedingte Verzweigung. Adreßdistanzwert 8 oder 16 Bit.
DBcc	Test der Bedingung, Dekrementierung und Verzweigung, Adreßdistanzwert 16 Bit.
Scc	Test der Bedingung und Setzen des Bytes auf FF oder 00.
BRA	Verzweigung. Adreßdistanzwert 8 oder 16 Bit.
BSR	Verzweigung ins Unterprogramm. Adreßdistanzwert 8 oder 16 Bit.
JMP	Sprung.
JSR	Sprung ins Unterprogramm.
RTR	Rückkehr aus Unterprogramm mit Wiederherstellung des Statusregisters vom Stapel.
RTS	Rückkehr aus Unterprogramm.

Systemsteuerbefehle

Befehl	Operation
Privilegierte Befehle	
ANDI to SR	Logisches UND mit Statusregister.
EORI to SR	Logisches EXKLUSIV-ODER mit Statusregister.
ORI to SR	Logisches ODER mit Statusregister.
MOVE EA to SR	Laden des Statusregisters.
MOVE USP	Datentransport von/nach Benutzerstapelzeiger.
RESET	Initialisierung der externen Schaltkreise.
RTE	Rückkehr von einer Ausnahme.
STOP	Anhalten der Programmausführung.
TRAP-Befehle	
CHK	Register auf Grenzen prüfen.
TRAP	Übergang in den Ausnahmezustand.
TRAPV	Übergang in den Ausnahmezustand, wenn V = 1.

Abb. 5.1: Der Befehlssatz, gegliedert nach Typen (Forts.)

Systemsteuerbefehle

Befehl	Operation
Bedingungscoderegister	
ANDI to CCR	Logisches UND mit dem Anwender-Statusregister.
EORI to CCR	Logisches EXKLUSIV-ODER mit dem Anwender-Statusregister.
ORI to CCR	Logisches ODER mit dem Anwender-Statusregister.
MOVE EA to CCR	Laden des Anwender-Statusregisters.
MOVE SR to EA	Speicherung des Statusregisters.

Abb. 5.1: Der Befehlssatz, gegliedert nach Typen (Forts.)

Während der Ausarbeitung der kleinen Anwendungsprogramme benötigen wir gewisse Assembleranweisungen oder „Pseudooperationen", die kleinere Verwaltungsaufgaben übernehmen können. Sie definieren Symbole, weisen Adressen für temporäre Speicherung zu, steuern Druckformate etc. Deshalb wollen wir uns diese Anweisungen zuerst anschauen.

ORG.S

legt die Anfangsadresse des Objektprogramms absolut kurz fest.

ORG.L

legt die Anfangsadresse des Objektprogramms absolut lang fest, d. h. man hat die Möglichkeit, mit 24-Bit-Adressen zu operieren. Wird keine Angabe gemacht, so ist .L standardmäßig festgelegt.

Beispiele:

```
                00002000                   ORG.L  $2000
00002000        4EF900003000               JMP    ADR1
WARNING 551
00002006        4EF900020000               JMP    ADR2
                00002100                   ORG    $2100
00002100        4EF900003000               JMP    ADR1
WARNING 551
00002106        4EF900020000               JMP    ADR2
                00003000                   ORG    $3000
00003000        4E71                 ADR1  NOP
                00020000                   ORG    $20000
00020000        4E71                 ADR2  NOP
                                           END
TOTAL ERRORS                    0
TOTAL WARNINGS                  2
```

Die beiden Warnungen weisen darauf hin, daß ADRI, das im Bereich ORG.L liegt, eine kurze Adresse ist.

```
                00002000                 ORG.L   $2000
00002000        4EF900003000             JMP     ADR1
WARNING 551
00002006        4EF900020000             JMP     ADR2
                00002100                 ORG.S   $2100
00002100        4EF83000                 JMP     ADR1
00002104        4AFB4E71                 JMP     ADR2
ERROR      252
                00003000                 ORG     $3000
00003000        4E71               ADR1  NOP
                00020000                 ORG     $20000
00020000        4E71               ADR2  NOP
                                         END
TOTAL ERRORS                 1
TOTAL WARNINGS               1
```

Der Fehler (ERROR) wird durch den Befehl JMP ADR2 in dem Bereich von ORG.S verursacht, da die Adresse ADR2 lang ist. Anstelle des Codes für JMP finden wir den Code für den illegalen Befehl 4AFB.

RORG

weist den relativen Speicherplatz zu.

Beispiel:

RORG $3000

Das Programm beginnt an der Adresse $3000, und die relativen Adressierungsarten sind erlaubt.

DC Define Constant

DC.B

definiert eine oder mehrere Speicherkonstanten mit Bytelänge, die im Speicher an aufeinanderfolgenden Adressen abgelegt werden.

Beispiele:

DC.B 'ABCDEF' Die 6 ASCII-Codes der Zeichen A bis F werden im Speicher abgelegt.

DC.B 'DAS PROGRAMM IST BEENDET.'

DC.B $0A,$08,$0B,$07

DC.W

definiert eine oder mehere Speicherkonstanten mit Wortlänge, die im Speicher an aufeinanderfolgenden Adressen abgelegt werden.

Beispiele:

NUL DC.W 0
VAL DC 'F' Ablegen der Worte 00, 0F.

DC.L

definiert eine oder mehrere Speicherkonstanten mit Doppelwortlänge, die im Speicher an aufeinanderfolgenden Adressen abgelegt werden.

Beispiel:

WERT DC.L $FFFFEE00

DS

DS.B n

reserviert einen Speicherbereich in der Länge von n Bytes.

Beispiel:

TABL DS.B 30 Reservierung von 30 Bytes ab der Adresse TABL.

DS.W n

reserviert einen Speicherbereich in der Länge von n Worten.

Beispiel:

WERT DS.W 4 Reservierung von 4 Worten ab der Adresse WERT.

DS.L n

reserviert einen Speicherbereich in der Länge von n Doppelworten.

Beispiel:

ZEIGER DS.L 2 Reservierung von 2 Doppelworten ab der Adresse ZEIGER.

SET

weist den Symbolen im Programm temporäre Werte zu.

Beispiel:

```
INDEX SET INDEX+1
```

EQU

setzt den Wert eines Symbols gleich einem anderen bis zu einer maximalen Länge von 32 Bits.

Beispiel:

VEKTBERR EQU 2*4	Die Ausnahmeverarbeitung Bus Error findet ihren Ausnahmevektor an der Adresse 8.
VEKTGL EQU 24*4	

Beispiel zur Verdeutlichung des Unterschieds von SET und EQU:

```
                00002000              ORG     $2000
                00000010        VAL1  EQU     $10
ERROR 234
                0000FFFF        VAL2  SET     $FFFF
00002000        4E71                  NOP
00002002        4E71                  NOP
00002004        4E71                  NOP
                00000020        VAL1  EQU     $20
ERROR 234
                0000EEEE        VAL2  SET     $EEEE
00002006        4E71                  NOP
                                      END
TOTAL ERRORS              2
TOTAL WARNINGS            0
```

Das Programm ruft zwei Assemblierungsfehler hervor, da der Wert WERT1 zweimal definiert ist, dies bei der Assembleranweisung EQU aber nicht erlaubt ist.

Dieses Problem stellt sich nicht für den Wert WERT2, da dieser durch SET definiert wurde.

END

ist die Assembleranweisung für das Programmende.

MACRO

definiert einen Makrobefehl.

Beispiel:

```
MAKPRO MACRO PROGA,PROGB
```

ENDM

zeigt das Ende des Makrobefehls an.

IFEQ xx

assembliert, wenn der angegebene Wert xx gleich 0 ist.

IFNE xx

assembliert, wenn der angegebene Wert xx gleich 1 ist.

ENDC

beendet die bedingte Assemblierung.

Beispiel:

Im folgenden Programm wird gemäß der Definition von SINN eine Links- oder Rechtsrotation ausgeführt.

```
SINN    EQU     1 oder 0
        ORG     $2000
        IFEQ    SINN
        ROL     #2,D0
        ENDC
        IFNE    SINN
        ROR     #3,D0
        ENDC
        END
```

Zum Zeitpunkt der Assemblierung erhält man entweder:

```
00000001   SINN   EQU    1
00002000          ORG    $2000
                  IFEQ   SINN
                  ENDC
                  IFNE   SINN
00002000   E658   ROR    #3, D0
                  ENDC
                  END
TOTAL ERRORS             0
TOTAL WARNINGS           0
```

oder:

```
00000000   SINN   EQU    0
00002000          ORG    $2000
                  IFEQ   SINN
00002000   E558   ROL    #2, D0
                  ENDC
                  IFNE   SINN
                  ENDC
                  END
TOTAL ERRORS             0
TOTAL WARNINGS           0
```

LLEN

bestimmt die Anzahl der Zeichen pro Zeile.

Beispiel:

LLEN 120 setzt 120 Zeichen pro Zeile.

PLEN

bestimmt die Anzahl der Zeilen pro Seite.

Beispiel:

PLEN 50 setzt 50 Zeilen pro Seite.

NOOBJ

bewirkt keine Ausgabe der Objektdatei.

SPC

bewirkt einen Zeilenvorschub.

Beispiel:

SPC 10 gibt 10 Zeilen Vorschub.

TTL

ermöglicht, eine maximal 60 Zeichen lange Programmüberschrift anzugeben.

BESCHREIBUNG DES BEFEHLSSATZES

Verwendete Abkürzungen:

An	–	Adreßregister n
D8	–	8-Bit-Adreßdistanzwert
D16	–	16-Bit-Adreßdistanzwert
Dn	–	Datenregister n
EA	–	effektive Adresse
PC	–	Programmzähler
SR	–	Statusregister
()	–	Inhalt von
<>	–	obligatorische Angaben
→	–	ergibt

Die Auswirkung der Befehlsausführung auf die Bedingungscodes wird durch folgende Notation angegeben:

B	Bit wird beeinflußt
U	undefiniert
–	Bit wird nicht beeinflußt
0	Bit wird auf 0 gesetzt
1	Bit wird auf 1 gesetzt

Befehle, die mit einem Stern * gekennzeichnet sind, werden in einem weiteren Abschnitt des Kapitels ausführlich mit Beispielen besprochen.

ABCD * Add Binary Coded Decimal
Addiere dezimal mit Erweiterungsbit

Operation: $(\text{Quelle})_{10}+(\text{Ziel})_{10}+X \rightarrow \text{Ziel}$

Assemblersyntax: ABCD Dy,Dx
oder
ABCD −(Ay),−(Ax)

Operandenlänge: Byte

Beschreibung: Die Addition wird mit binär codierten Dezimalzahlen durchgeführt, die sich entweder in zwei Datenregistern oder in zwei Speicherstellen befinden, wobei dann die Operanden in der Adressierungsart „Adreßregister indirekt mit Predekrementierung" adressiert werden. Dabei werden die im Befehl angegebenen Adreßregister verwendet.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0		Rx		1	0	0	0	0	R/M		Ry	

Bezeichnungen:

R/M=0 – Datenregister
R/M=1 – Adreßregister (Adreßregister indirekt mit Predekrementierung)
Rx – Zielregister
Ry – Quellregister

Bedingungscodes:

X	N	Z	V	C
B	U	B	U	B

N undefiniert.
Z wird auf 0 gesetzt, wenn das Ergebnis nicht 0 ist. Ist es 0, bleibt Z unverändert.
V undefiniert.

C wird auf 1 gesetzt, wenn ein dezimaler Übertrag angefallen ist, sonst wird es auf 0 zurückgesetzt.
X wie C.

Bei mehrfach genauen Operationen ist es erforderlich, Z vor der Befehlsdurchführung auf 1 zu setzen (siehe auch Programm 7 im Kapitel „Anwenderprogramme“).

ADD

Add Binary
Binäre Addition

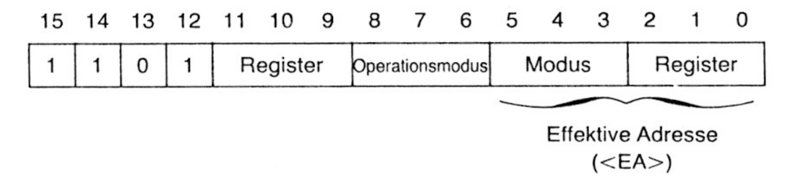
Operation: (Quelle)+(Ziel)→Ziel

Assemblersyntax: ADD <EA>,Dn
oder
ADD Dn,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Binäre Addition eines Operanden zu dem Inhalt eines Datenregisters, wobei der Zieloperand anschließend mit dem Ergebnis überschrieben wird.

Befehlsformat:



Bezeichnungen:

Operationsmodus

Byte	Wort	Doppelwort	Operation
000	001	010	(<Dn>)+(<EA>)→<Dn>
100	101	110	(<EA>+(<Dn>)→<EA>

Register – gibt eins der 8 Datenregister an.

Wenn <EA> die Quelle ist, sind folgende Adressierungsarten erlaubt:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An *	001	Nummer des Registers	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
−(An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

* Nur Wort und Doppelwort

Wenn <EA> das Ziel ist, sind folgende Adressierungsarten gestattet:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
−(An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird bei Überlauf auf 1 gesetzt, sonst auf 0.
C wird auf 1 gesetzt, wenn ein dezimaler Übertrag angefallen ist, sonst auf 0.
X wie C.

ADDA *Add Address* *Addiere Adresse*

Operation: (Quelle)+(Ziel)→Ziel

Assemblersyntax: ADD <EA>,An

Operandenlänge: Wort, Doppelwort

Beschreibung: Binäre Addition eines Operanden zu dem Inhalt eines Adreßregisters, wobei das Ergebnis im Adreßregister abgelegt wird.

Befehlsformat:

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	1	0	1	Register	Operationsmodus	Modus	Register

Effektive Adresse (Bits 5–0)

Bezeichnungen:

Register – gibt eins der 8 Adreßregister an.

Operationsmodus – 011 – Wortoperation. Der Quelloperand wird dabei auf 32 Bits erweitert.
111 – Doppelwortoperation.

<EA> ist immer die Quelle, alle Adressierungsarten sind erlaubt:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	001	Nummer des Registers	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
−(An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

Bedingungscodes: unverändert.

ADDI *Add Immediate* *Addiere unmittelbar*

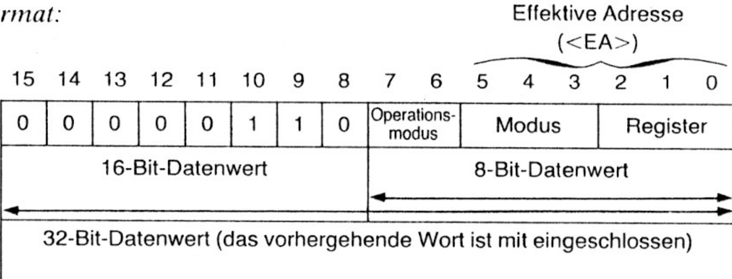
Operation: <unmittelbares Datenelement>+ (Ziel)→Ziel

Assemblersyntax: ADDI #<Datenelement>,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Binäre Addition eines unmittelbaren Datenelementes zu einem Zieloperanden, wobei der Zieloperand mit dem Ergebnis überschrieben wird.

Befehlsformat:



Bezeichnungen:

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort

<EA> ist immer das Ziel. Folgende Addressierungsarten sind erlaubt:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An)+	011	Nummer des Registers	d(PC)	–	–
–(An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird auf 1 gesetzt, wenn ein Überlauf entsteht, sonst auf 0.
C wird auf 1 gesetzt, wenn ein Übertrag anfällt, sonst auf 0.
X wie C.

Wenn die Addition mit einem Adreßregister durchgeführt wird, werden die Bedingungscodes nicht berührt.

ADDQ *Add Quick* *Addiere schnell*

Operation: <unmittelbares Datenelement>+(Ziel)→Ziel

Assemblersyntax: ADDQ #<Datenelement>,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Binäre Addition eines unmittelbaren Datenelementes zu einem Zieloperanden, wobei der Zieloperand mit dem Ergebnis überschrieben wird. Die Daten können die Werte 0 bis 7 (000 bis 111) annehmen.

Befehlsformat:

15	14	13	12	11 10 9	8	7 6	5 4 3	2 1 0
0	1	0	1	Daten	0	Operations-modus	Modus	Register

Effektive Adresse (Modus + Register)

Bezeichnungen:

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort

<EA> ist immer das Ziel. Folgende Addressierungsarten sind erlaubt:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An*	001	Nummer des Registers	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
–(An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

* Nur Wort und Doppelwort

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird auf 1 gesetzt, wenn ein Überlauf entsteht, sonst auf 0.
C wird auf 1 gesetzt, wenn ein Übertrag anfällt, sonst auf 0.
X wie C.

Wenn die Addition mit einem Adreßregister durchgeführt wird, werden die Bedingungscodes nicht berührt.



ADDX **Add with Extend** / **Addiere mit Erweiterungsbit**

Operation: (Quelle)+(Ziel)+X→Ziel

Assemblersyntax: ADDX Dy,Dx
oder
ADDX −(Ay),−(Ax)

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Addition des Quell- zu dem Zieloperanden und dem Erweiterungsbit, wobei der Zieloperand mit dem Ergebnis überschrieben wird. Die Operanden sind entweder in Datenregistern oder in Speicherfeldern enthalten.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Register Rx			1	Operations-modus		0	0	R/M	Register Ry		

Bezeichnungen:

R/M=0 – Datenregister.
R/M=1 – Adreßregister mit Predekrementierung.
Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 0 gesetzt, wenn das Ergebnis ungleich 0 ist, sonst auf unverändert.

V wird auf 1 gesetzt, wenn ein Überlauf entsteht, sonst auf 0.
C wird auf 1 gesetzt, wenn ein Übertrag anfällt, sonst auf 0.
X wie C.

Bei Operationen mit mehrfacher Genauigkeit ist es ratsam, vor der Ausführung Z auf 1 zu setzen (siehe auch Programm 7 im Kapitel „Anwenderprogramme").



AND Logical
Logisches UND

Operation: (Quelle) ∧ (Ziel) → Ziel

Assemblersyntax: AND <EA>,Dn
oder
AND Dn,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Der Quelloperand wird mit dem Zieloperanden über ein logisches UND verknüpft. Dabei darf der Operand nicht Inhalt eines Adreßregisters sein.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register			Operations-modus			Modus			Register		

Effektive Adresse (Bits 5–0)

Bezeichnungen:

Register – gibt eins der 8 Datenregister an.

Operationsmodus

Byte	Wort	Doppelwort	Operation
000	001	010	(Dn)UND(<EA>)→Dn
100	101	110	(<EA>)UND(Dn) →<EA>

Wenn <EA> die Quelle ist, sind folgende Adressierungsarten erlaubt:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
–(An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

Wenn <EA> das Ziel ist, sind folgende Adressierungsarten gestattet:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
–(An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das höchstwertige Bit des Ergebnisses 1 ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird immer auf 0 gesetzt.
X unbeeinflußt.

ANDI

AND Immediate
Logisches UND unmittelbar

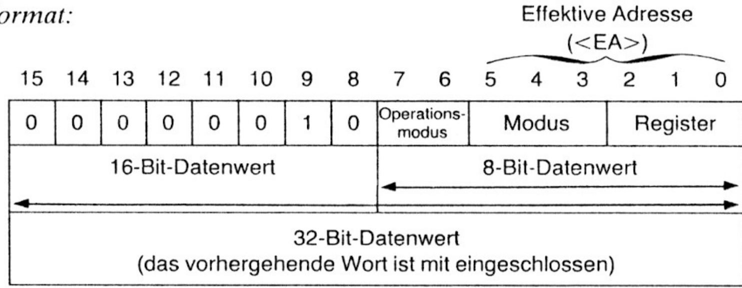
Operation: <unmittelbare Daten> ∧ (Ziel) → Ziel

Assemblersyntax: ANDI #<Daten>,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Die unmittelbaren Daten werden mit dem Zieloperanden über ein logisches UND verknüpft.

Befehlsformat:



Bezeichnungen:

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort

<EA> ist der Quelloperand, und folgende Adressierungsarten sind erlaubt:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
–(An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das höchstwertige Bit des Ergebnisses 1 ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird immer auf 0 gesetzt.
X unbeeinflußt.

ANDI mit CCR

AND Immediate to Condition Code Register
Logisches UND unmittelbar mit Bedingungscoderegister

Operation: <unmittelbare Daten> ∧ (Bedingungscoderegister) → Bedingungscoderegister

Assemblersyntax: ANDI #<8-Bit-Daten>,CCR

Operandenlänge: Byte

Beschreibung: Die unmittelbaren Daten von 8 Bit Länge werden mit dem Bedingungscoderegister über ein logisches UND verknüpft.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	8-Bit-Datenwert							

Bedingungscodes: Die Bedingungscodes werden entsprechend der logischen Verknüpfung beeinflußt.

Beispiel:

Das Bit C wird auf 0 gesetzt, wenn das Bit 0 der unmittelbaren Daten 0 ist; andernfalls bleibt es unverändert. Dasselbe gilt für N, Z, V, X.

ANDI mit SR

AND Immediate to Status Register
Logisches UND unmittelbar mit vollem Statusregister

Operation: Wenn der Prozessor im Supervisor-Modus ist:

<unmittelbare Daten> $\wedge$ SR $\rightarrow$ SR

andernfalls Ausnahmeverarbeitung „Privilegverletzung".

Assemblersyntax: ANDI #<16-Bit-Daten>,SR

Operandenlänge: Wort

Beschreibung: Die unmittelbaren Daten von 16 Bit Länge werden mit dem Statusregister SR über ein logisches UND verknüpft, wenn der Prozessor im Supervisor-Modus ist, andernfalls liegt eine Privilegverletzung vor.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	1	1	1	1	1	0	0
unmittelbarer 16-Bit-Datenwert															

Bedingungscodes: Die Bedingungscodes werden entsprechend der logischen Verknüpfung beeinflußt, und zwar nur durch die 5 niederwertigsten Bits der unmittelbaren Daten.

Beispiel:

Das Bit N wird auf 0 gesetzt, wenn das Bit 3 der unmittelbaren Daten 0 ist; andernfalls bleibt es unverändert. Dasselbe gilt für Z, V, C, X.

ASL, ASR * *Arithmetic Shift Left, Right* *Arithmetische Verschiebung links, rechts*

Operation: (Ziel) verschoben um<Stellenzahl>→Ziel

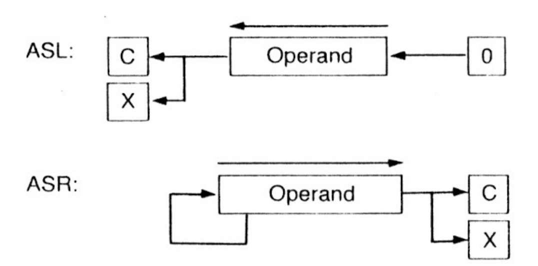
Assemblersyntax:
ASL Dx,Dy ; ASR Dx,Dy
ASL #<Daten>,Dn ; ASR #<Daten>,Dn
ASL <EA> ; ASR <EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Die Bits des Operanden werden in der angegebenen Richtung arithmetisch verschoben. Bei einer Registerverschiebung kann der Verschiebungsfaktor auf zwei verschiedene Weisen angegeben werden:

– durch einen unmittelbaren Wert (von 1 bis 8)
– durch einen Wert in einem Datenregister

Der Inhalt einer Speicherstelle kann nur um ein Bit verschoben werden, und der Operand muß immer ein Wort sein.



Befehlsformat: Verschiebung eines Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Anzahl/ Register			dr	Operations-modus		i/r	0	0	Register		

Bezeichnungen:

Anzahl/Register – bestimmt die Anzahl der Verschiebungen oder das Register, das diese Zahl enthält.

i/r – i/r=0 – die Anzahl der Verschiebungen (von 1 bis 8) ist im Befehl enthalten.
i/r=1 – die Anzahl der Verschiebungen (modulo 64) ist im angegebenen Register enthalten.

dr – gibt die Richtung der Verschiebung an.
dr=0 – Verschiebung nach rechts.
dr=1 – Verschiebung nach links.

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort

Register – gibt die Nummer des Datenregisters an, dessen Inhalt verschoben werden soll.

Befehlsformat: Verschiebung einer Speicherstelle

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	dr	1	1	Modus			Register		

Effektive Adresse
<EA>

Bezeichnungen:

dr – bestimmt die Richtung der Verschiebung.
dr=0 – Verschiebung nach rechts.
dr=1 – Verschiebung nach links.

Effektive Adresse – bestimmt den Operanden, der verschoben werden soll.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

N wird auf 1 gesetzt, wenn das höchstwertige Bit des Ergebnisses 1 ist, sonst auf 0.

Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.

V wird auf 1 gesetzt, wenn das höchstwertige Bit während der Verschiebung verändert wird, sonst auf 0.

C wird entsprechend dem letzten aus dem Operanden geschobenen Bit gesetzt. Es wird auf 0 gesetzt, wenn keine Verschiebung stattgefunden hat.

X wird wie C entsprechend dem letzten aus dem Operanden geschobenen Bit gesetzt. Es bleibt unverändert, wenn keine Verschiebung stattgefunden hat.

Bcc ∗ *Branch Conditionally* *Bedingte Verzweigung*

Operation: Wenn (Bedingung erfüllt) dann PC+D→PC

Assemblersyntax: Bcc <Label>

Operandenlänge: Byte, Wort

Beschreibung: Wenn die angegebene Bedingung erfüllt ist, wird die Verarbeitung des Programms an der Adresse PC+ Adreßdistanzwert fortgesetzt.

Der Wert von PC ist die Adresse des aktuellen Befehls plus 2.

Der Adreßdistanzwert enthält die Differenz in Bytes zwischen der Adresse des Bcc-Befehls und dem angegebenen Label. Ist der Distanzwert ein 8-Bit-Wert, dann ist er im Befehl selbst enthalten. Ist er ein 16-Bit-Wert, so sind die 8 Bits des Befehls 0, und der Adreßdistanzwert ist im folgenden Wort enthalten.

Die Bedingungen cc sind im einzelnen:

0100	CC	Carry Clear	kein Übertrag	$C = 0$
0101	CS	Carry Set	Übertrag	$C = 1$
0111	EQ	EQual	gleich	$Z = 1$
0110	NE	Not Equal	ungleich	$Z = 0$
1100	GE	Greater or Equal	größer oder gleich	$N \oplus V = 0$
1110	GT	Greater Than	größer	$Z + (N \oplus V) = 0$
0010	HI	HIgh	größer (ohne Vorzeichen)	$C + Z = 0$
1111	LE	Less or Equal	kleiner oder gleich	$Z + (N \oplus V) = 1$
0011	LS	Less or Same	kleiner oder gleich (ohne Vorzeichen)	$C + Z = 1$
1101	LT	Less Than	kleiner	$N \oplus V = 1$
1011	MI	MInus	negativ	$N = 1$
1010	PL	PLus	positiv	$N = 0$
1000	VC	oVerflow Clear	kein Überlauf	$V = 0$
1001	VS	oVerflow Set	Überlauf	$V = 1$
0000	T	True	wahr	1
0001	F	False	falsch	0

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	Bedingung				8-Bit-Adreßdistanzwert							
16-Bit-Adreßdistanzwert, wenn 8-Bit-Adreßdistanzwert Null ist															

Bedingungscodes: nicht beeinflußt.

BCHG * *Bit Test and Change* *Ändern eines Bits nach Test*

Operation: Bit-Nummer vom Ziel→Z
Bit-Nummer vom Ziel→logisches Komplement

Assemblersyntax: BCHG Dn,<EA>
oder
BCHG #<Daten>,<EA>

Operandenlänge: Byte, Doppelwort

Beschreibung: Ein Bit des Zieloperanden wird getestet, und sein Wert bestimmt den Bedingungscode Z. Anschließend wird das Bit komplementiert. Der Zieloperand kann ein Datenregister sein. In diesem Fall wird eins der 32 Bits des Registers getestet. Falls der Zieloperand in einer Speicherstelle steht, so wird eins der 8 Bits des Bytes, das sich an dieser Stelle befindet, getestet. Die Nummer des zu testenden Bits wird entweder unmittelbar oder in einem Datenregister festgelegt.

Befehlsformat: BCHG Dn,<EA>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Register (Dn)			1	0	1	Modus			Register		
										Effektive Adresse					

Bezeichnungen:

Register (Dn) – ist das Datenregister, das die Bit-Nummer enthält.

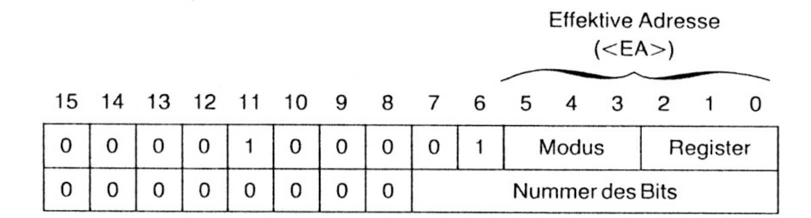
Effektive Adresse – gibt den Zieloperanden an.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn*	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
–(An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

* Nur Doppelwort. Bei den anderen Modi nur Byte.

Befehlsformat: BCHG #<Daten>,<EA>



Bedingungscodes:

X	N	Z	V	C
–	–	B	–	–

N nicht beeinflußt.
Z wird auf 1 gesetzt, wenn das getestete Bit 0 ist, sonst auf 0.
V nicht beeinflußt.
C nicht beeinflußt.
X nicht beeinflußt.

BCLR ✳ *Bit Test and Clear*
Löschen eines Bits nach Test

Operation: Bit-Nummer xxx des Ziels→Z
0→Bit-Nummer xxx des Ziels

Assemblersyntax: BCLR Dn,<EA>
oder
BCLR #<Daten>,<EA>

Operandenlänge: Byte, Doppelwort

Beschreibung: Ein Bit des Zieloperanden wird getestet, und sein Wert bestimmt den Bedingungscode Z. Anschließend wird das Bit gelöscht. Der Zieloperand kann ein Datenregister sein. In diesem Fall wird eins der 32 Bits des Registers getestet. Falls der Zieloperand in einer Speicherstelle steht, so wird eins der 8 Bits des Bytes, das sich an dieser Stelle befindet, getestet. Die Nummer des zu testenden Bits wird entweder unmittelbar oder in einem Datenregister festgelegt.

Befehlsformat: BCLR Dn,<EA>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Register			1	1	0	Modus			Register		

Effekive Adresse

Bezeichnungen:

Register – gibt die Nummer des Registers Dn an, das die Nummer des auf 0 zu setzenden und zu testenden Bits enthält.
Effektive Adresse – bestimmt den Zieloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn*	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

* Nur Doppelwort. Bei den anderen Modi nur Byte.

Befehlsformat: BCLR #<Daten>,<EA>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										Effektive Adresse					
0	0	0	0	1	0	0	0	1	0	Modus			Register		
0	0	0	0	0	0	0	0	Nummer des Registers							

Bezeichnungen:

Effektive Adresse – bestimmt den Zieloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn*	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

* Nur Doppelwort. Bei den anderen Modi nur Byte.

Bedingungscodes:

X	N	Z	V	C
–	–	B	–	–

N nicht beeinflußt.
Z wird auf 1 gesetzt, wenn das getestete Bit 0 ist, sonst auf 0.
V nicht beeinflußt.
C nicht beeinflußt.
X nicht beeinflußt.



BRA

Branch Always
Unbedingte Verzweigung

Operation: PC+D→PC

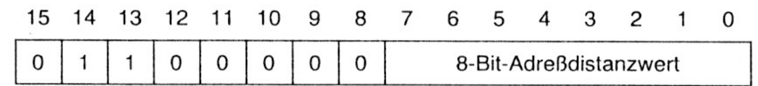
Assemblersyntax: BRA <Label>

Operandenlänge: Byte, Wort

Beschreibung: Der nächste auszuführende Befehl steht an der Adresse PC+Adreßdistanzwert. Der Distanzwert ergibt sich aus der Differenz zwischen dem Wert des PC im Augenblick der Ausführung des Befehls BRA ((PC)+2) und dem angegebenen Label.

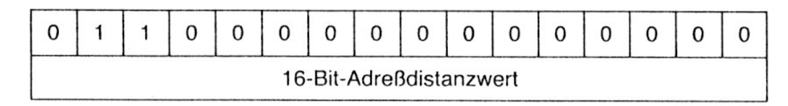
Ist der Distanzwert ein 8-Bit-Wert, so ist er im Befehlswort enthalten. Ist er ein 16-Bit-Wert, so ist er im Erweiterungswort enthalten, und die 8 Bits des Befehlswortes sind auf 0 gesetzt. Der Adreßdistanzwert wird immer im Zweierkomplement angegeben.

Befehlsformat:



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	8-Bit-Adreßdistanzwert							

oder aber:



0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
16-Bit-Adreßdistanzwert															

Bedingungscodes: nicht beeinflußt.

BSET ✳ *Bit Test and Set* *Setze Bit nach Test*

Operation: Bit-Nummer xxx des Ziels→Z
1→Bit-Nummer xxx des Ziels

Assemblersyntax: BSET Dn,<EA>
oder
BSET #<Daten>,<EA>

Operandenlänge: Byte, Doppelwort

Beschreibung: Ein Bit des Zieloperanden wird getestet, und sein Wert bestimmt den Bedingungscode Z. Danach wird dieses Bit im Zieloperanden auf 1 gesetzt. Das Ziel kann ein Datenregister sein, dabei ist das zu testende Bit eins der 32 Bits des Registers. Wenn das Ziel eine Speicherstelle ist, ist das zu testende Bit eins der 8 Bits des Bytes, das sich an dieser Stelle befindet. Die Bitnummer ist entweder unmittelbar angegeben oder in einem Datenregister enthalten.

Befehlsformat: BSET Dn,<EA>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Register			1	1	1	Modus			Register		

Effektive Adresse (Bits 5–0)

Bezeichnungen:

Register – bezeichnet das Datenregister, das die Bit-Nummer angibt.

Effektive Adresse – legt den Zieloperanden fest.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn*	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

* Nur Doppelwort. Bei den anderen Modi nur Byte.

Befehlsformat: BSET #<Daten>,<EA>

Effektive Adresse (Bits 5–0)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	1	Modus			Register		
0	0	0	0	0	0	0	0	Nummer des Bits							

Bezeichnungen:

Effektive Adresse – legt den Zieloperanden fest.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn*	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

* Nur Doppelwort. Bei den anderen Modi nur Byte.

Bedingungscodes:

X	N	Z	V	C
–	–	B	–	–

N nicht beeinflußt.
Z wird auf 1 gesetzt, wenn das getestete Bit 0 ist, sonst auf 0.
V nicht beeinflußt.
C nicht beeinflußt.
X nicht beeinflußt.

BSR

Branch to Subroutine
Verzweigung zum Unterprogramm

Operation: PC→–(SP)
PC+D→PC

Assemblersyntax: BSR <Label>

Operandenlänge: Byte, Wort

Beschreibung: Die 32-Bit-Adresse des Befehls, die auf den Befehl BSR folgt, wird auf den Stapel gerettet. Das Programm wird an der Adresse (PC)+Adreßdistanzwert weiter ausgeführt. Die Distanz stellt den relativen Abstand zwischen dem Wert des Programmzählers und dem Label dar. Dieser Distanzwert wird durch Bildung des ganzzahligen Zweierkomplements ermittelt und entweder als 8- oder 16-Bit-Zahl angegeben.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8-Bit-Adreßdistanzwert							
16-Bit-Adreßdistanzwert (wenn 8-Bit-Adreßdistanzwert Null ist)															

Bedingungscodes: nicht beeinflußt.

BTST *

Bit Test
Teste Bit

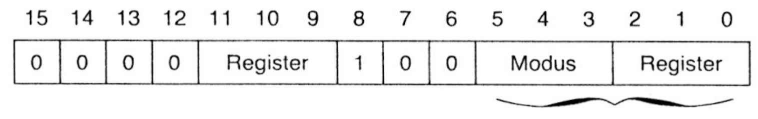
Operation: Bit-Nummer xxx des Ziels→Z

Assemblersyntax: BTST Dn,<EA>
oder
BTST #<Daten>,<EA>

Operandenlänge: Byte, Doppelwort

Beschreibung: Ein Bit des Zieloperanden wird getestet, und sein Wert bestimmt das Bit Z. Das Ziel kann ein Datenregister sein, dabei ist das zu testende Bit eins der 32 Bits des Registers. Wenn das Ziel eine Speicherstelle ist, ist das zu testende Bit eins der 8 Bits des Bytes, das sich an dieser Stelle befindet. Die Bit-Nummer ist entweder unmittelbar angegeben oder in einem Datenregister enthalten.

Befehlsformat: BTST Dn,<EA>



Bezeichnungen:

Register – gibt das Datenregister an, das die Bit-Nummer enthält.

Effektive Adresse – legt den Zieloperanden fest.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn*	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	–	111	–
– (An)	100	Nummer des Registers	–	111	–
d(An)	101	Nummer des Registers	–	111	–

* Nur Doppelwort. Bei den anderen Modi nur Byte.

Befehlsformat: BTST #<Daten>,<EA>

Effektive Adresse (Bits 5–0)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	Modus			Register		
0	0	0	0	0	0	0	0	Nummer des Bits							

Bezeichnungen:
Effektive Adresse – legt den Zieloperanden fest.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn*	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

* Nur Doppelwort. Bei den anderen Modi nur Byte.

Bedingungscodes:

X	N	Z	V	C
–	–	B	–	–

N nicht beeinflußt.
Z wird auf 1 gesetzt, wenn das getestete Bit 0 ist, sonst auf 0.
V nicht beeinflußt.
C nicht beeinflußt.
X nicht beeinflußt.

CHK * *Check Registers Against Bounds* *Prüfe Register auf Grenzen*

Operation: Wenn Dn<0 oder Dn>(<EA>), dann TRAP, sonst nächster Befehl

Assemblersyntax: CHK <EA>,Dn

Operandenlänge: Wort

Beschreibung: Die 16 niederwertigen Bits des angegebenen Datenregisters werden mit der oberen Grenze (behandelt als Zweierkomplement) des mit der effektiven Adresse angesprochenen Operanden verglichen. Wenn der Wert des Registers kleiner als 0 oder größer als diese obere Grenze ist, dann geht der Prozessor in den Ausnahmezustand CHK, und es wird die Vektornummer 6 erzeugt.

Befehlsformat:

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td colspan="3">Register</td><td>1</td><td>1</td><td>0</td><td colspan="3">Modus</td><td colspan="3">Register</td></tr>
<tr><td colspan="10"></td><td colspan="6">Effektive Adresse</td></tr>
</table>

Bezeichnungen:

Register – gibt das Datenregister an, dessen Inhalt verglichen wird.
Effektive Adresse – gibt die obere Grenze an.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
– (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

Bedingungscodes:

X	N	Z	V	C
–	B	U	U	U

N wird auf 1 gesetzt, wenn Dn kleiner als 0 ist. Es wird auf 0 gestzt, wenn Dn größer ist als (<EA>), ansonsten ist es undefiniert.
Z undefiniert.
V undefiniert.
C undefiniert.
X nicht beeinflußt.

CLR

Clear Operand
Setze Operand auf 0

Operation: 0→Ziel

Assemblersyntax: CLR <EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Alle Bits des Zieloperanden werden auf 0 gesetzt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	Operations-modus		Modus			Register		

Effektive Adresse (Modus, Register)

Bezeichnungen:

Operationsmodus – 00 – Byte
01 – Wort
10 – Datenwort

Effektive Adresse – legt den Zieloperanden fest.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
–	0	1	0	0

N wird auf 0 gesetzt
Z wird auf 1 gesetzt.
V wird auf 0 gesetzt.
C wird auf 0 gesetzt.
X nicht beeinflußt.



CMP

Compare
Vergleiche

Operation: (<Dn>)−(<EA>)
(Ziel)−(Quelle)

Assemblersyntax: CMP <EA>,Dn

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Der Quelloperand wird dem Zieloperanden subtrahiert. Die Bedingungscodes werden dem Ergebnis entsprechend gesetzt. Der Zieloperand bleibt unverändert.

Befehlsformat:

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	1	1	Register	Operationsmodus	Modus	Register
						Effektive Adresse	

Bezeichnungen:

Register – gibt das Ziel-Datenregister an.
Operationsmodus – 000 – Byte
001 – Wort
010 – Doppelwort
Effektive Adresse – bestimmt den Quelloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An*	001	Nummer des Registers	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
− (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

* Nur Wort und Doppelwort

Bedingungscodes:

X	N	Z	V	C
–	B	B	B	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird auf 1 gesetzt, wenn ein Überlauf anfällt, sonst auf 0.
C wird auf 1 gesetzt, wenn es einen Übertrag gibt, sonst auf 0.
X nicht beeinflußt.

CMPA *Compare Address* *Vergleiche Adresse*

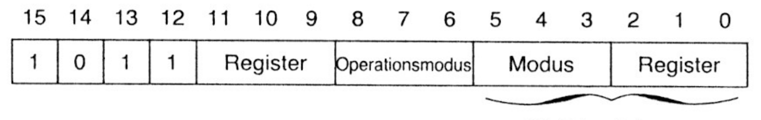
Operation: (<An>)−(<EA>)
(Ziel)−(Quelle)

Assemblersyntax: CMPA <EA>,An

Operandenlänge: Wort, Doppelwort

Beschreibung: Der Quelloperand wird dem Zieloperanden subtrahiert. Die Bedingungscodes werden dem Ergebnis entsprechend gesetzt. Das Register An bleibt unverändert. Bei einer Wortoperation wird der Quelloperand auf 32 Bits erweitert.

Befehlsformat:



Bezeichnungen:

Register – gibt das Ziel-Adreßregister an.

Operationsmodus – 011 – Wortoperation mit Erweiterung des Quelloperanden)
111 – Doppelwortoperation

Effektive Adresse – bestimmt den Quelloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	001	Nummer des Registers	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
– (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

Bedingungscodes:

X	N	Z	V	C
–	B	B	B	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird auf 1 gesetzt, wenn ein Überlauf anfällt, sonst auf 0.
C wird auf 1 gesetzt, wenn es einen Übertrag gibt, sonst auf 0.
X nicht beeinflußt.

CMPI *Compare Immediate Vergleiche unmittelbar*

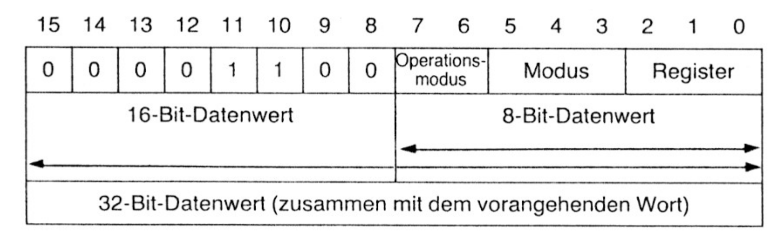
Operation: (<An>)−(<unmittelbare Daten>)
(Ziel)−(Quelle)

Assemblersyntax: CMPA #<Daten>,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Das unmittelbare Datenelement wird vom Zieloperanden subtrahiert. Die Bedingungscodes werden dem Ergebnis entsprechend gesetzt. Der Zieloperand bleibt unverändert. Die Länge des unmittelbaren Datenelements stimmt mit der Angabe im Datenlängencode überein.

Befehlsformat:



Bezeichnungen:

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort

Effektive Adresse – bestimmt den Zieloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
–	B	B	B	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird auf 1 gesetzt, wenn ein Überlauf anfällt, sonst auf 0.
C wird auf 1 gesetzt, wenn es einen Übertrag gibt, sonst auf 0.
X nicht beeinflußt.

CMPM *Compare Memory Vergleiche Speicher*

Operation: (Ziel)−(Quelle)

Assemblersyntax: CMPM (Ay)+,(Ax)+

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Der Quelloperand wird vom Zieloperanden subtrahiert. Die Bedingungscodes werden dem Ergebnis entsprechend gesetzt. Die Operanden werden durch Postinkrementierung unter Zuhilfenahme der Adreßregister angesprochen.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	Register Ax			1	Operationsmodus		0	0	1	Register Ay		

Bezeichnungen:

Register Ax − bestimmt den Zieloperanden (für die Postinkrementierung).

Register Ay − bestimmt den Quelloperanden.

Operationsmodus − 00 − Byte
01 − Wort
10 − Doppelwort

Bedingungscodes:

X	N	Z	V	C
−	B	B	B	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.

Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird auf 1 gesetzt, wenn ein Überlauf anfällt, sonst auf 0.
C wird auf 1 gesetzt, wenn es einen Übertrag gibt, sonst auf 0.
X nicht beeinflußt.

DBcc

Test, Decrement and Branch
Prüfe, dekrementiere und verzweige

Operation: Wenn die Bedingung nicht erfüllt ist, dann:
Dn−1→Dn
Wenn Dn≠−1, dann (PC)+D→PC,
sonst (PC)+2→PC
Wenn die Bedingung erfüllt ist, dann keine Operation

Assemblersyntax: DBcc Dn,<Label>

Operandenlänge: Wort

Beschreibung: Dieser Befehl hat 3 Parameter: eine Bedingung, ein Datenregister und einen Adreßdistanzwert. Er kann für die Implementierung von Schleifen verwendet werden. Dabei wird die Bedingung cc getestet. Wenn die Abbruchbedingung cc nicht erfüllt ist, werden die unteren 16 Bits des Registers Dn um 1 vermindert. Wenn dabei −1 herauskommt, wird der nächstfolgende Befehl ausgeführt. Wenn das Ergebnis ungleich −1 ist, wird die Programmausführung an der durch den Wert des PC und den Adreßdistanzwert (erweitert) festgelegten Stelle fortgesetzt. Die Bedingungen cc können die folgenden sein:

0100	CC	Carry Clear	kein Übertrag	$C=0$
0101	CS	Carry Set	Übertrag	$C=1$
0111	EQ	EQual	gleich	$Z=1$
0110	NE	Not Equal	ungleich	$Z=0$
1100	GE	Greater or Equal	größer oder gleich	$N \oplus V=0$
1110	GT	Greater Than	größer	$Z+(N \oplus V)=0$
0010	HI	HIgh	größer (ohne Vorzeichen)	$C+Z=0$
1111	LE	Less or Equal	kleiner oder gleich	$Z+(N \oplus V)=1$
0011	LS	Less or Same	kleiner oder gleich (ohne Vorzeichen)	$C+Z=1$
1101	LT	Less Than	kleiner	$N \oplus V=1$
1011	MI	MInus	negativ	$N=1$
1010	PL	Plus	positiv	$N=0$
1000	VC	oVerflow Clear	kein Überlauf	$V=0$
1001	VS	oVerflow Set	Überlauf	$V=1$
0000	T	True	wahr	1
0001	F	False	falsch	0

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Bedingung				1	1	0	0	1	Register		
Adreßdistanzwert															

Bezeichnungen:

Bedingung – enthält eine der 16 in der Tabelle angegebenen Bedingungen.

Register – gibt das Datenregiser an, das als Zähler verwendet wird.

Adreßdistanzwert – gibt den Abstand zwischen dem ausgeführten Befehl und dem angesprochenen Label an.

Bedingungscodes: unbeeinflußt.

DIVS * *Signed Divide* *Division mit Vorzeichen*

Operation: (Ziel)/(Quelle)→Ziel

Assemblersyntax: DIVS <EA>,Dn

Operandenlänge: Wort

Beschreibung: Der 32-Bit-Zieloperand wird durch den 16-Bit-Quelloperanden dividiert. Die Operation wird entsprechend der arithmetischen Vorzeichen durchgeführt. Das 32-Bit-Ergebnis wird folgendermaßen abgespeichert: der Quotient im unteren Wort, der Rest im oberen Teil des Wortes. Das Vorzeichen des Restes entspricht dem des Dividenden, außer wenn er 0 ist. Zwei Sonderfälle können vorkommen: Bei Division durch 0 wird ein Ausnahmezustand ausgelöst (Vektor 5). Ein Überlauf wird vor dem Ende der Befehlsausführung entdeckt, und der entsprechende Bedingungscode wird gesetzt. In diesem Fall werden die Operanden nicht verändert. Es entsteht ein Überlauf, wenn der Quotient einschließlich Vorzeichen größer als 16 Bit ist.

Befehlsformat:

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	0	0	0	Register	1	1	1	Modus	Register

Effektive Adresse (Modus, Register)

Bezeichnungen:

Register – gibt das Datenregister an, das den Zieloperanden enthält.

Effektive Adresse – gibt den Quelloperanden an.

Die zugelassenen Adressierungsarten sind die folgenden:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
– (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

Bedingungscodes:

X	N	Z	V	C
–	B	B	B	0

N wird auf 1 gesetzt, wenn der Quotient negativ ist, sonst auf 0. Es ist undefiniert, wenn V=1 ist.
Z wird auf 1 gesetzt, wenn der Quotient 0 ist, sonst auf 0. Es ist undefiniert, wenn V=1 ist.
V wird auf 1 gesetzt, wenn ein Überlauf anfällt, sonst auf 0.
C wird auf 0 gesetzt.
X nicht beeinflußt.

DIVU * *Unsigned Divide* *Division ohne Vorzeichen*

Operation: (Ziel)/(Quelle)→Ziel

Assemblersyntax: DIVU <EA>,Dn

Operandenlänge: Wort

Beschreibung: Die Division erfolgt ohne Vorzeichen. Sonst sind die Merkmale dieses Befehls dieselben wie die von DIVS. Man muß nur beachten, daß das Bedingungscodebit N auf 1 gesetzt wird, wenn das höchstwertige Bit des Quotienten gesetzt ist, andernfalls wird N auf 0 gesetzt. Wenn ein Überlauf auftritt und V also 1 wird, ist N undefiniert. Ein Überlauf tritt dann auf, wenn der Quotient ohne Vorzeichen größer als 16 Bits wird.

Befehlsformat:

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	0	0	0	Register	0	1	1	Modus	Register

Effektive Adresse

Bezeichnungen:

Register – gibt das Datenregister an, das den Zieloperanden enthält.

Effektive Adresse – gibt den Quelloperanden an.

Die zugelassenen Adressierungsarten sind die folgenden:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
– (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

Bedingungscodes:

X	N	Z	V	C
–	B	B	B	0

N wird auf 1 gesetzt, wenn das höchstwertige Bit des Quotienten 1 ist, sonst auf 0. Es ist undefiniert, wenn V=1 ist.
Z wird auf 1 gesetzt, wenn der Quotient 0 ist, sonst auf 0. Es ist undefiniert, wenn V=1 ist.
V wird auf 1 gesetzt, wenn ein Überlauf anfällt, sonst auf 0.
C wird auf 0 gesetzt.
X nicht beeinflußt.

EOR

Exclusive OR Logical
Logisches exclusives ODER

Operation: $(\text{Quelle}) \veebar (\text{Ziel}) \rightarrow \text{Ziel}$

Assemblersyntax: EOR Dn,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Der Quelloperand wird mit dem Zieloperanden über ein logisches exklusives ODER verknüpft, und das Ergebnis wird in den Zieloperanden geschrieben.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	Register			Operationsmodus			Modus			Register		

Effektive Adresse

Bezeichnungen:

Register – gibt das Datenregister an, (Quelloperand).
Operationsmodus – 100 – Byte
101 – Wort
110 – Doppelwort
Effektive Adresse – gibt den Zieloperanden an.

Die zugelassenen Adressierungsarten sind die folgenden:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das höchstwertige Bit des Ergebnisses 1 ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird auf 0 gesetzt.
C wird auf 0 gesetzt.
X nicht beeinflußt.

EORI

Exclusive OR Immediate
Exklusives ODER unmittelbar

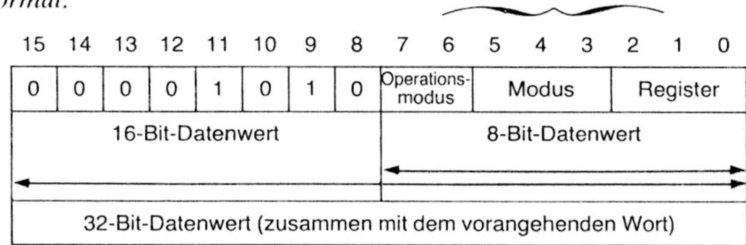
Operation: <unmittelbare Daten>∨(Ziel)→Ziel

Assemblersyntax: EORI #<Daten>,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Das unmittelbare Datenelement wird mit dem Zieloperanden über ein exklusives ODER verknüpft. Die Länge des unmittelbaren Datenelements entspricht dem angegebenen Datenlängencode.

Befehlsformat:



Bezeichnungen:

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort

Effektive Adresse – gibt den Zieloperanden an.

Die zugelassenen Adressierungsarten sind die folgenden:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das höchstwertige Bit des Ergebnisses 1 ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird auf 0 gesetzt.
C wird auf 0 gesetzt.
X nicht beeinflußt.

EORI mit CCR

Exclusive OR Immediate to CCR
Exklusives ODER unmittelbar mit CCR

Operation: <unmittelbare Daten>⊻CCR→CCR

Assemblersyntax: EORI #<8-Bit-Daten>,CCR

Operandenlänge: Byte

Beschreibung: Das unmittelbare 8-Bit-Datenelement wird mit dem unteren Byte des Statusregisters SR über ein exklusives ODER verknüpft.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	8-Bit-Datenwert							

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

Alle Bedingungscodes werden entsprechend der logischen Verknüpfung beeinflußt.

Beispiel: X wird gesetzt, wenn das Bit 4 des Datenelements 1 ist und X vorher 0 war.

EORI mit SR *Exclusive OR Immediate to SR* *Exklusives ODER unmittelbar mit SR*

Operation: Im Supervisor-Modus:
<unmittelbare Daten>⊻SR→SR
Sonst Ausnahmezustand „Privilegverletzung" (Vektor 8).

Assemblersyntax: EORI #<16-Bit-Daten>,SR

Operandenlänge: Wort

Beschreibung: Wenn der Prozessor im Supervisor-Modus ist, wird das unmitelbare 16-Bit-Datenelement mit dem Statusregister SR über ein exklusives ODER verknüpft. Andernfalls wird der Ausnahmezustand „Privilegverletzung" hervorgerufen.

Befehlsformat:

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td colspan="16">16-Bit-Datenwert</td></tr>
</table>

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

Alle Bedingungscodes werden entsprechend der logischen Verknüpfung beeinflußt.

Beispiel: N wird gesetzt, wenn das Bit 3 des Datenelements 1 ist und N vorher 0 war.

EXG

Exchange Registers
Datenaustausch zwischen Registern

Operation: Rx↔Ry

Assemblersyntax: EXG Rx,Ry

Operandenlänge: Doppelwort

Beschreibung: Die 32-Bit-Inhalte von zwei Registern werden miteinander vertauscht. Dieser Austausch kann stattfinden zwischen:

- Datenregistern
- Adreßregistern
- einem Daten- und einem Adreßregister

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register Rx			1	Operationsmodus					Register Ry		

Bezeichnungen:

Register Rx – gibt das Daten- oder Adreßregister an. Im Fall des Daten-/Adreßregisteraustauschs gibt es das Datenregister an.

Operationsmodus – 01000 – Datenregister
01001 – Adreßregister
10001 – Daten- und Adreßregister

Register Ry – gibt das Daten- oder Adreßregister an. Im Fall des Daten-/Adreßregisteraustauschs gibt es das Adreßregister an.

Bedingungscodes: nicht beeinflußt.

EXT * *Sign Extend* *Vorzeichenerweiterung*

Operation: (Ziel)vorzeichenerweitert→Ziel

Assemblersyntax: EXT Dn

Operandenlänge: Wort, Doppelwort

Beschreibung:

Wortoperation – Bit 7 wird in die Bits 8 bis 15 kopiert.

Doppelwort-operation – Bit 15 wird in die Bits 16 bis 31 kopiert.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	Operations-modus			0	0	0	Register		

Bezeichnungen:

Operationsmodus – 010 – Erweiterung von Byte-Wortlänge.
011 – Erweiterung von Wort-auf Doppelwortlänge.

Register – gibt das Datenregister an, dessen Inhalt erweitert wird.

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird auf 0 gesetzt.
C wird auf 0 gesetzt.
X nicht beeinflußt.

ILLEGAL

Illegal Instruction
Unzulässiger Befehl

Operation: (PC)→−(SSP); (SR)→−(SSP)
(Vektor 4)→PC

Assemblersyntax: –

Operandenlänge: nicht definiert.

Beschreibung: Der Befehl, der den Hexadezimalcode $4AFC besitzt, erzeugt die Ausnahmebedingung „illegale Operation", der den Ausnahmevektor 4 hat.

Befehlsformat: durch die Beschreibung festgelegt.

Bedingungscodes: nicht beeinflußt.

JMP

Jump
Sprung

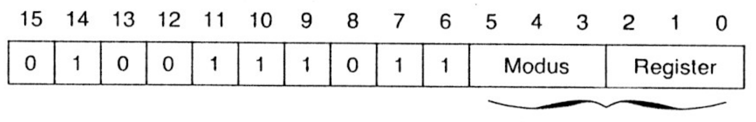
Operation: Ziel→PC

Assemblersyntax: JMP <EA>

Operandenlänge: –

Beschreibung: Die Ausführung des Programms wird an der im Befehl angegebenen Adresse fortgesetzt.

Befehlsformat:



Bezeichnungen:

Effektive Adresse – bestimmt die Adresse des nächsten auszuführenden Befehls.

Folgende Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	–	–	d(PC)	111	010
– (An)	–	–	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes: nicht beeinflußt.

JSR

Jump to Subroutine
Sprung zum Unterprogramm

Operation: PC→–(SP); Ziel→PC

Assemblersyntax: JSR <EA>

Operandenlänge: –

Beschreibung: Die Adresse des unmittelbar auf JSR folgenden Befehls wird auf den Stapel gerettet. Danach wird die Ausführung des Programms an der durch JSR festgelegten Adresse fortgesetzt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
0	1	0	0	1	1	1	0	1	0	Modus	Register

Effektive Adresse (Modus, Register)

Bezeichnungen:

Effektive Adresse – bestimmt die Adresse des nächsten auszuführenden Befehls.

Folgende Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	–	–	d(PC)	111	010
– (An)	–	–	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes: nicht beeinflußt.

LEA * *Load effective Address* *Laden der effektiven Adresse*

Operation: Ziel→An

Assemblersyntax: LEA <EA>,An

Operandenlänge: Doppelwort

Beschreibung: Die effektive Adresse wird in das angegebene Adreßregister geladen. Alle 32 Bits dieses Registers sind davon betroffen.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	Register			1	1	1	Modus			Register		

Effektive Adresse (Bits 5–0)

Bezeichnungen:

Register – legt das Adreßregister fest, in das die effektive Adresse geladen wird.

Effektive Adresse – bestimmt die zu ladende Adresse.

Folgende Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	–	–	d(PC)	111	010
– (An)	–	–	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes: nicht beeinflußt.

LINK * *Link and Allocate* *Verbinden und Speicher reservieren*

Operation: An→–(SP); SP→An; SP+D→SP

Assemblersyntax: LINK An,#<16-Bit-Adreßdistanzwert>

Operandenlänge: –

Beschreibung: Der Inhalt des angegebenen Adreßregisters wird auf den Stapel gerettet. Danach wird der Inhalt des Stapelzeigers SP in das Adreßregister geladen. Anschließend wird der auf 32 Bit erweiterte Distanzwert zum SP addiert. Über den negativen Distanzwert ist es möglich, gezielt Stapelfelder freizuzgeben.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	Register		
Adreßdistanzwert															

Bezeichnungen:

Register – gibt das Adreßregister an, über das die Verbindung hergestellt wird.

Adreßdistanzwert – wird als Zweiterkomplement gebildet und zum Stapelzeiger hinzuaddiert.

Bedingungscodes: nicht beeinflußt.

LSL, LSR * *Logical Shift Left, Right* *Logische Verschiebung links, rechts*

Operation: (Ziel) verschoben um <Stellenzahl>→Ziel

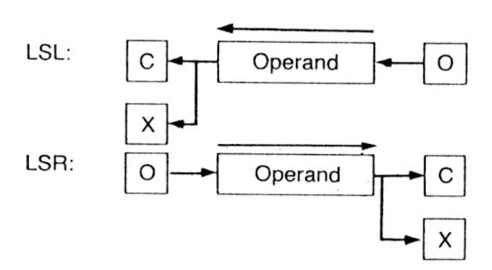
Assemblersyntax:
LSL Dx,Dy ; LSR Dx,Dy
LSL #<Daten>,Dn; LSR #<Daten>,Dn
LSL <EA> ; LSR <EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Die Bits des Operanden werden in der angegebenen Richtung logisch verschoben. Bei einer Registerverschiebung kann der Verschiebungsfaktor auf zwei verschiedene Weisen angegeben werden:

- durch einen unmittelbaren Wert (von 1 bis 8)
- durch einen Wert in einem Datenregister

Der Inhalt einer Speicherstelle kann nur um ein Bit verschoben werden, und der Operand muß immer ein Wort sein.



Befehlsformat: Verschiebung eines Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Anzahl/ Register			dr	Operationsmodus		i/r	0	1	Register		

Bezeichnungen:

Anzahl/Register	–	bestimmt die Anzahl der Verschiebungen oder das Register, das diese Zahl enthält.
i/r	–	i/r=0 – der Verschiebungsfaktor (von 1 bis 8) ist im Befehl enthalten. i/r=1 – der Verschiebungsfaktor (modulo 64) ist im angegebenen Register enthalten.
dr	–	gibt die Richtung der Verschiebung an. dr=0 – Verschiebung nach rechts. dr=1 – Verschiebung nach links.
Operationsmodus	–	00 – Byte 01 – Wort 10 – Doppelwort
Register	–	gibt die Nummer des Datenregisters an, dessen Inhalt verschoben werden soll.

Befehlsformat: Verschiebung einer Speicherstelle

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
1	1	1	0	0	0	1	dr	1	1	Modus	Register

Effektive Adresse

Bezeichnungen:

dr	–	bestimmt die Richtung der Verschiebung. dr=0 – Verschiebung nach rechts. dr=1 – Verschiebung nach links.
Effektive Adresse	–	bestimmt den Operanden, der verschoben werden soll.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
B	B	B	0	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird durch das letzte aus dem Operanden geschobene Bit bestimmt. Es wird auf 0 gesetzt, wenn keine Verschiebung stattgefunden hat.
X wird wie C durch das letzte aus dem Operanden geschobene Bit bestimmt. Es wird nicht beeinflußt, wenn keine Verschiebung stattgefunden hat.

MOVE *Move Data from Source to Destination* *Transportiere Daten von der Quelle zum Ziel*

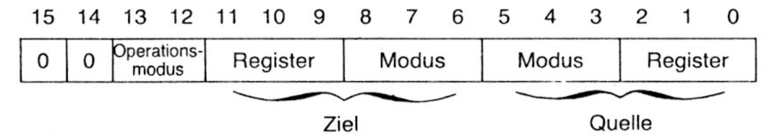
Operation: (Quelle)→Ziel

Assemblersyntax: MOVE <EA>,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Der Quelloperand wird zum Ziel transportiert. Die Daten werden bei der Übertragung getestet, und dementsprechend werden die Bedingungscodes gesetzt.

Befehlsformat:



Bezeichnungen:

Operationsmodus – 01 – Byte
11 – Wort
10 – Doppelwort
Ziel – bestimmt das Ziel.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Quelle – bestimmt den zu übertragenden Quelloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An *	001	Nummer des Registers	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
– (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

* Für die Byteoperationen ist die Adressierungsart Adreßregister direkt verboten

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird immer auf 0 gesetzt.
X nicht beeinflußt.

MOVE zum CCR

Move to CCR
Transportiere Daten zum CCR

Operation: (Quelle)→CCR

Assemblersyntax: MOVE <EA>,CCR

Operandenlänge: Wort

Beschreibung: Der Quelloperand wird zum Bedingungscoderegister transportiert. Der Operand ist zwar ein Wort, aber nur die 8 unteren Bits beeinflussen das CCR, die 8 höheren Bits werden ignoriert.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	Modus			Register		

Effektive Adresse

Bezeichnungen:

Effektive Adresse – bestimmt den Quelloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
– (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

Bemerkung: MOVE zum CCR ist eine auf ein Wort erweiterte Operation. AND, OR und EOR mit CCR sind Operationen, die sich auf ein Byte beziehen.

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

N wird entsprechend dem Bit 3 des Quelloperanden gesetzt.
Z wird entsprechend dem Bit 2 des Quelloperanden gesetzt.
V wird entsprechend dem Bit 1 des Quelloperanden gesetzt.
C wird entsprechend dem Bit 0 des Quelloperanden gesetzt.
X wird entsprechend dem Bit 4 des Quelloperanden gesetzt.

MOVE zum SR *Move to SR* *Transportiere Daten zum SR*

Operation: Wenn der Prozessor im Supervisor-Modus ist, dann:

(Quelle)→SR

Sonst Ausnahmezustand „Privilegverletzung".

Assemblersyntax: MOVE <EA>,SR

Operandenlänge: Wort

Beschreibung: Der Quelloperand wird nur zum Statusregister SR übertragen, wenn sich der Prozessor im Supervisor-Modus befindet. Andernfalls wird der Ausnahmezustand „Privilegverletzung" ausgelöst.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
0	1	0	0	0	1	1	0	1	1	Modus	Register

Effektive Adresse

Bezeichnungen:

Effektive Adresse – bestimmt den Quelloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
– (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

Die Bedingungscodes werden dem Quelloperanden entsprechend gesetzt.

MOVE von SR *Move from SR* *Transportiere Daten vom SR*

Operation: SR→Ziel

Assemblersyntax: MOVE SR,<EA>

Operandenlänge: Wort

Beschreibung: Der Inhalt des Statusregisters SR wird zur Zieladresse übertragen.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	Modus			Register		

Effektive Adresse

Bezeichnungen:

Effektive Adresse – bestimmt den Zieloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes: nicht beeinflußt.

MOVE USP

Move User Stack Pointer
Transportiere Daten von und nach Benutzerstapelzeiger

Operation: Wenn der Prozessor im Supervisor-Modus ist, dann:
USP→An oder An→USP

Sonst Ausnahmezustand „Privilegverletzung".

Assemblersyntax: MOVE USP,An
oder
MOVE An,USP

Operandenlänge: Doppelwort

Beschreibung: Der Inhalt des Benutzerstapelzeigers wird vom oder zum angegebenen Adreßregister übertragen.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	dr	Register		

Bezeichnungen:

dr – gibt die Übertragungsrichtung an.
dr=0 – An→USP
dr=1 – USP→An

Register – gibt das Adreßregister an, das beim Transfer verwendet wird.

Bedingungscodes: nicht beeinflußt.

MOVEA *Move Address* *Transportiere Adresse*

Operation: (Quelle)→Ziel

Assemblersyntax: MOVEA <EA>,An

Operandenlänge: Wort, Doppelwort

Beschreibung: Der Quelloperand wird zum Adreßregister übertragen. Operanden der Länge 16 werden vor der Übertragung vorzeichenbehaftet auf 32 Bits erweitert.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	Operations-modus		Register			0	0	1	Modus			Register		
				Ziel						Quelle					

Bezeichnungen:

Operationsmodus – gibt die Länge des zu übertragenden Operanden an.
11 – Wortoperation mit Erweiterung auf 32 Bits.
10 – Doppelwortoperation

Ziel – gibt das verwendete Adreßregister an.

Quelle – gibt den Quelloperanden an.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	001	Nummer des Registers	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
– (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

Bedingungscodes: nicht beeinflußt.

MOVEM * *Move Multiple Registers* *Transportiere mehrere Register*

Operation: mehrere Register→Ziel
oder
(Quelle→mehrere Register

Assemblersyntax: MOVEM <Registerliste>,<EA>
oder
MOVEM <EA>,<Registerliste>

Operandenlänge: Wort, Doppelwort

Beschreibung: Die angegebenen Register werden von oder zum Speicher transportiert, beginnend mit derjenigen Speicherstelle, die durch die effektive Adresse bestimmt wird. Das Erweiterungswort des Befehls wird als Registermaske bezeichnet. Wenn ein Register von dem Befehl betroffen werden soll, wird das entsprechende Bit in der Registermaske auf 1 gesetzt. Werden nun die Worte zu den Registern übertragen, erfolgt eine vorzeichenbehaftete Erweiterung auf 32 Bits, wobei dies für die Adreßregister sowie für die Datenregister gilt. MOVEM erlaubt die drei folgenden Adressierungsarten:

- Adreßregister indirekt mit Predekrementierung.
- Adreßregister indirekt mit Postinkrementierung.
- Steuerung (alle Adressierungsarten zum Zugriff zu Speicheroperanden ohne Größenangaben).

Bei der Adressierung mit Predekrementierung werden die Register in der Reihenfolge Adreßregister A7 bis A0 und danach Datenregister D7 bis D0 übertragen. Für die beiden anderen Adressierungsarten geschieht der Transport in der umgekehrten Reihenfolge, und zwar werden zuerst die Register D0 bis D7 und dann A0 bis A7 übertragen.

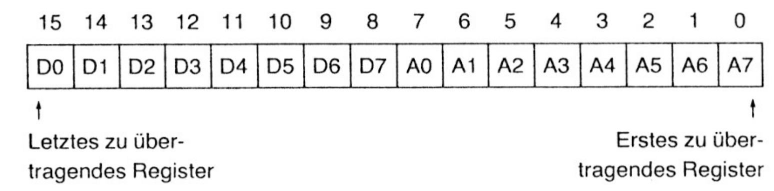
Transport vom Register zum Speicher

Folgende Adressierungsarten sind zulässig:

- Adreßregister indirekt :(An)

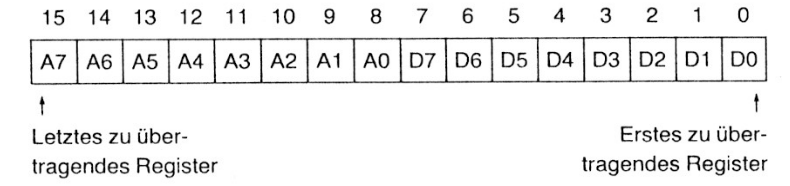
– Predekrementierung :–(An)
– indirekt mit Adreßdistanz:D(An)
– indirekt mit Index :D(An,XI)
– absolut kurz und lang :Abs.W,Abs.L

Im Fall der Predekrementierung hat die Registermaske folgende Struktur:



Im Speicher werden die Register in der Reihenfolge der abnehmenden Adressen (Predekrementierung) übertragen. Das erste Register wird an der effektiven Adresse minus 2 (Predekrementierung für Worte) geschrieben. Durch die Registermaske ist genau festgelegt, welche Register betroffen sind.

Im Fall der Adressierungsart Steuerung hat die Registermaske folgende Struktur:



Die Register werden zu der angegebenen Adresse in der Reihenfolge der aufsteigenden Adressen übertragen.

Transport vom Speicher zum Register

Folgende Adressierungsarten sind zulässig:

– Adreßregister indirekt :(An)
– Predekrementierung :(An)+
– indirekt mit Adreßdistanz:D(An)
– indirekt mit Index :D(An,XI)

– absolut kurz und lang :Abs.W,Abs.L
– relativ zum PC :D(PC)
– relativ zum PC mit Index :D(PC,XI)

Im Fall der Postinkrementierung:

Die Register werden mit dem Inhalt der angegebenen Speicherstellen geladen. Danach wird der Inhalt der nächsten Speicherfelder in der Reihenfolge der aufsteigenden Adressen übertragen.

Für den Fall der Postinkrementierung und der Adressierungsart Steuerung hat die Registermaske folgende Struktur:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										Effektive Adresse					
0	1	0	0	1	dr	0	0	1	Operationsmodus	Modus			Register		
Registermaske															

Bezeichnungen:

dr – gibt die Transportrichtung an.
dr=0 – Register zum Speicher
dr=1 – Speicher zum Register

Operationsmodus – gibt die Länge der für den Transport benutzten Register an.
0 – Wort
1 – Doppelwort

Effektive Adresse – gibt die Speicheradresse an, zu oder ab der ein Transport durchgeführt wird.

Für den Fall des Transportes von Registern zum Speicher sind folgende Adressierungsarten zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	–	–	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Für den Fall des Transportes vom Speicher zu den Registern sind folgende Adressierungsarten zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
– (An)	–	–	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes: nicht beeinflußt.

MOVEP * *Move Peripheral Data* *Übertrage periphere Daten*

Operation: (Quelle)→Ziel

Assemblersyntax: MOVEP Dx,D(Ay)
oder
MOVEP D(Ay),Dx

Operandenlänge: Wort, Doppelwort

Beschreibung: Die Daten werden zwischen den Datenregistern und abwechselnd aufeinanderfolgenden geraden oder aufeinanderfolgenden ungeraden Speicherstellen übertragen. Daher erfolgt die Adreßinkrementierung mit der Schrittweite 2. Das obere Byte eines Datenregisters wird als erstes übertragen, das untere als letztes. Die Speicheradresse wird stets in der Art „indirekt mit Adreßdistanzwert" vorgegeben. Bei einer geraden Adresse werden alle Übertragungen auf der oberen Hälfte des Datenbusses durchgeführt, bei einer ungeraden auf der unteren Hälfte.

Im folgenden Beispiel wird die Übertragung eines Doppelwortes von oder zu einer geraden Speicheradresse veranschaulicht:

Byteorganisation des Registers:

31 … 24	23 … 16	15 … 8	7 … 0
höchstwertiges Byte	zweithöchstwertiges Byte	zweitniederwertigstes Byte	niederwertigstes Byte

Byteorganisation des Speichers:

aufsteigende Adressen ↓	15 … 8	7 … 0
	höchstwertiges Byte	
	zweithöchstwertiges Byte	
	zweitniederwertigstes Byte	
	niederwertigstes Byte	

Im folgenden Beispiel wird die Übertragung eines Wortes von oder zu einer ungeraden Speicheradresse veranschaulicht:

Byteorganisation des Registers:

31 … 24	23 … 16	15 … 8	7 … 0
		höherwertiges Byte	niederwertiges Byte

Byteorganisation des Speichers:

aufsteigende Adressen ↓

15 … 8	7 … 0
	höherwertiges Byte
	niederwertiges Byte

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Daten-register			Operations-modus			0	0	1	Adreß-register		
Adreßdistanzwert															

Bezeichnungen:

Datenregister – gibt das Datenregister an, von oder zu dem Daten übertragen werden.

Operationsmodus – gibt die Übertragungsrichtung und -länge an.
100 – Wort: Transport vom Speicher zu Registern.
101 – Doppelwort: Transport vom Speicher zu Registern.
110 – Wort: Transport von den Registern zum Speicher.
111 – Doppelwort: Transport von den Registern zum Speicher.

Adreßregister – bestimmt das Adreßregister, das bei der Adressierung mit Distanzwert verwendet wird.

Adreßdistanzwert – gibt die Verschiebung an, die bei der Berechnung der Operandenadresse Verwendung findet.

Bedingungscodes: nicht beeinflußt.

MOVEQ * *Move Quick* *Übertrage schnell*

Operation: <unmitttelbare Daten>→,Dn

Assemblersyntax: MOVEQ #<Daten>Ziel

Operandenlänge: Doppelwort

Beschreibung: Ein unmittelbares Datenelement wird in ein Datenregister übertragen. Es ist, mit 8 Bit codiert, direkt im Befehlswort enthalten und wird vor der Befehlsausführung auf 32 Bits vorzeichenbehaftet erweitert.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	Register			0	Datenwert							

Bezeichnungen:

Register – bestimmt das als Ziel verwendete Datenregister.

Datenwert – steht für das 8 Bits lange Datenelement, das vor dem Transfer auf 32 Bits erweitert wird.

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird immer auf 0 gesetzt.
X nicht beeinflußt.

MULS *Signed Multiply* *Multiplikation mit Vorzeichen*

Operation: (Quelle)*(Ziel)→Ziel

Assemblersyntax: MULS <EA>,Dn

Operandenlänge: Wort

Beschreibung: Zwei 16-Bit-Operanden mit Vorzeichen werden multipliziert, und der Zieloperand wird mit dem mit Vorzeichen versehenen 32-Bit-Ergebnis überschrieben. Von dem in einem Register enthaltenen Operanden werden nur die 16 unteren Bits berücksichtigt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register			1	1	1	Modus			Register		

Effektive Adresse (Bits 5–0)

Bezeichnungen:

Register – gibt das als Ziel verwendete Datenregister an.

Effektive Adresse – definiert den Quelloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
– (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird immer auf 0 gesetzt.
X nicht beeinflußt.

MULU *Unsigned Multiply* *Multiplikation ohne Vorzeichen*

Operation: (Quelle)∗(Ziel)→Ziel

Assemblersyntax: MULU <EA>,Dn

Operandenlänge: Wort

Beschreibung: Zwei 16-Bit-Operanden ohne Vorzeichen werden multipliziert, und das Datenregister des Zieloperanden wird mit dem 32-Bit-Ergebnis ohne Vorzeichen überschrieben. Von dem in einem Register enthaltenen Operanden werden nur die 16 unteren Bits berücksichtigt.

Befehlsformat:

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	1	0	0	Register	0	1	1	Modus	Register

Effektive Adresse (Bits 5–0)

Bezeichnungen:

Register – gibt das als Ziel verwendete Datenregister an.
Effektive Adresse – definiert den Quelloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
– (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird immer auf 0 gesetzt.
X nicht beeinflußt.

NBCD * *Negate Decimal with Extend* *Dezimale Negation mit Erweiterungsbit*

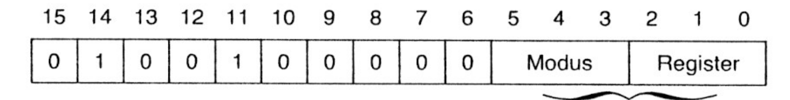
Operation: $0-(\text{Ziel})_{10}-X \rightarrow \text{Ziel}$

Assemblersyntax: NBCD <EA>

Operandenlänge: Byte

Beschreibung: Der Zieloperand und das Erweiterungsbit werden von 0 subtrahiert. Die Operation kann nur byteweise mit dezimaler Arithmetik durchgeführt werden und wird vorwiegend für BCD-Zahlen verwendet. Der Zieloperand wird mit dem Ergebnis überschrieben. Falls das Erweiterungsbit X gleich 0 ist, wird das Zehnerkomplement des Zieloperanden gebildet. Ist X gleich 1, dann wird ein Neunerkomplement verwendet.

Befehlsformat:



Bezeichnungen:

Effektive Adresse – definiert den Zieloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
B	U	B	U	B

N undefiniert.
Z wird auf 0 gesetzt, wenn das Ergebnis nicht 0 ist, sonst nicht beeinflußt.
V undefiniert.
C wird auf 1 gesetzt, wenn ein dezimaler Übertrag anfällt, sonst auf 0.
X wie C.

Bei mehrfach genauen Berechnungen ist es notwendig, daß das Bit Z vorher mit 1 initialisiert wird (siehe auch Programm 7 im Kapitel „Anwenderprogramme").

NEG

Negate
Negation

Operation: 0−(Ziel)→Ziel

Assemblersyntax: NEG <EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Der Zieloperand wird von 0 subtrahiert und vom Ergebnis überschrieben.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	Operationsmodus		Modus			Register		

Effektive Adresse

Bezeichnungen:

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort
Effektive Adresse – definiert den Zieloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn ein Überlauf anfällt, sonst auf 0.
V wird auf 1 gesetzt, wenn ein Überlauf anfällt, sonst auf 0.
C wird auf 1 gesetzt, wenn ein negativer Übertrag anfällt, sonst auf 0.
X wie C.

NEGX *Negate with Extend* *Negation mit Erweiterungsbit*

Operation: 0−(Ziel)−X→Ziel

Assemblersyntax: NEGX <EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Der Zieloperand und das Erweiterungsbit werden gemeinsam von 0 subtrahiert, und der Zieloperand wird vom Ergebnis überschrieben.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	Operations-modus		Modus			Register		

Effektive Adresse

Bezeichnungen:

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort
Effektive Adresse – definiert den Zieloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
B	U	B	U	B

N undefiniert.
Z wird auf 0 gesetzt, wenn das Ergebnis nicht 0 ist, sonst nicht beeinflußt.
V undefiniert.
C wird auf 1 gesetzt, wenn ein negativer Übertrag anfällt, sonst auf 0.
X wie C.

Bei mehrfach genauen Berechnungen, bei denen dieser Befehl vorzugsweise Verwendung findet, ist es notwendig, daß das Bit Z vorher mit 1 initialisiert wird (siehe auch Programm 7 im Kapitel „Anwenderprogramme").

NOP

No Operation
Keine Operation

Operation: –

Assemblersyntax: NOP

Operandenlänge: –

Beschreibung: Es wird keine Operation durchgeführt. Der Zustand des Prozessors bleibt unverändert. Der PC gibt an, welcher Befehl als nächster ausgeführt werden soll.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

Bedingungscodes: nicht beeinflußt.

NOT

Logical Complement
Einerkomplement

Operation: (Ziel)→Ziel

Assemblersyntax: NOT <EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Es wird das Einerkomplement des Zieloperanden gebildet.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	Operations-modus		Modus			Register		

Effektive Adresse (Modus, Register)

Bezeichnungen:

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort
Effektive Adresse – definiert den Zieloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird immer auf 0 gesetzt.
X nicht beeinflußt.

OR

Logical OR
Logisches ODER

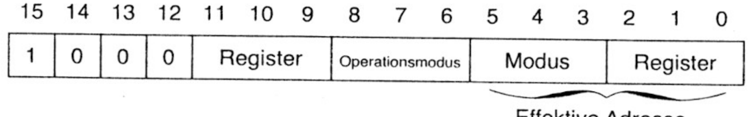
Operation: (Ziel)V(Quelle)→Ziel

Assemblersyntax: OR <EA>,Dn
oder
OR Dn,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Ziel- und Quelloperand werden durch ein logisches ODER verknüpft, und das Ergebnis überschreibt den Zieloperanden. Der Inhalt der Adreßregister darf nicht als Operand verwendet werden.

Befehlsformat:



Bezeichnungen:

Register – gibt das Datenregister an.
Operationsmodus

Byte	Wort	Doppelwort	Operation
000	001	010	<Dn>V<EA>→<Dn>
100	101	110	<EA>V<Dn>→<EA>

Effektive Adresse – definiert den Quelloperanden.

Dann sind die folgenden Adressierungsarten zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
– (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

Effektive Adresse – definiert den Zieloperanden.

Dann sind die folgenden Adressierungsarten zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
–(An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das höchstwertige Bit des Ergebnisses 1 ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird immer auf 0 gesetzt.
X nicht beeinflußt.

ORI

Logical OR Immediate
Logisches ODER unmittelbar

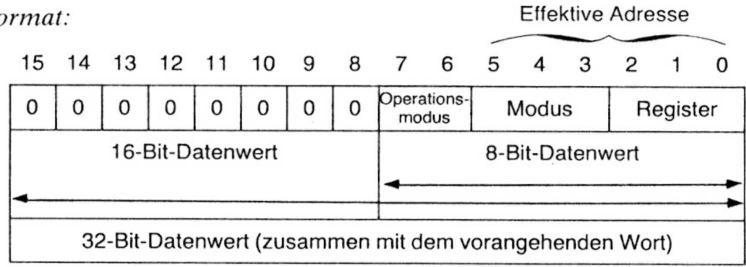
Operation: <unmittelbare Daten>V(Ziel)→Ziel

Assemblersyntax: ORI #<Daten>,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Das unmittelbare Datenelement und der Zieloperand werden durch ein logisches ODER verknüpft, und der Zieloperand wird mit dem Ergebnis überschrieben. Die Länge der unmittelbaren Daten stimmt mit dem Datenlängencode überein.

Befehlsformat:



Bezeichnungen:

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort

Effektive Adresse – definiert den Zieloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das höchstwertige Bit des Ergebnisses 1 ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird immer auf 0 gesetzt.
X nicht beeinflußt.

ORI mit CCR

Logical OR Immediate to CCR
Logisches ODER unmittelbar mit CCR

Operation: <unmittelbare Daten>VCCR→CCR

Assemblersyntax: ORI #<Daten>,CCR

Operandenlänge: Byte

Beschreibung: Das unmittelbare 8-Bit-Datenelement und das Bedingungscoderegister werden durch ein logisches ODER verknüpft. Das Ergebnis wird in das untere Byte des Statusregisters SR geschrieben.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	8-Bit-Datenwert							

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

N wird auf 1 gesetzt, wenn das Bit 3 des Datenelementes eine 1 enthält, sonst bleibt es unverändert.
Z wird auf 1 gesetzt, wenn das Bit 2 des Datenelementes eine 1 enthält, sonst bleibt es unverändert.
V wird auf 1 gesetzt, wenn das Bit 1 des Datenelementes eine 1 enthält, sonst bleibt es unverändert.
C wird auf 1 gesetzt, wenn das Bit 0 des Datenelementes eine 1 enthält, sonst bleibt es unverändert.
X wird auf 1 gesetzt, wenn das Bit 4 des Datenelementes eine 1 enthält, sonst bleibt es unverändert.

ORI mit SR

Logical OR Immediate to SR
Logisches ODER unmittelbar mit SR

Operation: Wenn der Prozessor im Supervisor-Modus ist, dann:

<unmittelbare Daten>∨SR→SR

Sonst tritt der Ausnahmezustand „Privilegverletzung“ auf.

Assemblersyntax: ORI #<16-Bit-Daten>,SR

Operandenlänge: Wort

Beschreibung: Das unmittelbare 8-Bit-Datenelement und das Statusregister SR werden durch ein logisches ODER verknüpft. Das Ergebnis wird in das Statusregister SR geschrieben. Wenn der Prozessor sich dabei nicht im Supervisor-Modus befindet, wird der Ausnahmezustand „Privilegverletzung“ mit der Vektornummer 8 erzeugt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
16-Bit-Datenwert															

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

N wird auf 1 gesetzt, wenn das Bit 3 des Datenelementes eine 1 enthält, sonst bleibt es unverändert.
Z wird auf 1 gesetzt, wenn das Bit 2 des Datenelementes eine 1 enthält, sonst bleibt es unverändert.
V wird auf 1 gesetzt, wenn das Bit 1 des Datenelementes eine 1 enthält, sonst bleibt es unverändert.
C wird auf 1 gesetzt, wenn das Bit 0 des Datenelementes eine 1 enthält, sonst bleibt es unverändert.
X wird auf 1 gesetzt, wenn das Bit 4 des Datenelementes eine 1 enthält, sonst bleibt es unverändert.

PEA * *Push Effective Address* *Ablegen der effektiven Adresse auf den Stapel*

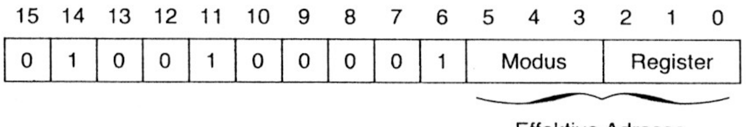
Operation: Ziel→–(SP)

Assemblersyntax: PEA <EA>

Operandenlänge: Doppelwort

Beschreibung: Die effektive Adresse wird berechnet und auf dem Stapel abgelegt. Dabei wird der Stapelzeiger SP um 4 vermindert.

Befehlsformat:



Bezeichnungen:

Effektive Adresse – gibt die auf den Stapel zu rettende Adresse an.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	–	–	d(PC)	111	010
– (An)	–	–	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes: nicht beeinflußt.

RESET

Reset External Devices
Zurücksetzen externer Einheiten in den Grundzustand

Operation: Wenn der Prozessor im Supervisor-Modus ist, dann wird der Reset-Ausgang entsprechend gesetzt, ansonsten geht der Prozessor in den Ausnahmezustand „Privilegverletzung“.

Assemblersyntax: RESET

Operandenlänge: –

Beschreibung: Die Reset-Leitung wird für die Dauer von 124 Taktzyklen aktiviert, um die externen Bausteine in den Grundzustand zurückzusetzen. Mit Ausnahme des PC, der den nächsten auszuführenden Befehl angibt, bleibt der Zustand des Prozessors unverändert.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

Bedingungscodes: nicht beeinflußt.

ROL, ROR *

Rotate Left, Right without Extend
Ringverschiebung links, rechts ohne Erweiterungsbit

Operation: (Ziel)ringverschoben um<Stellenzahl>→Ziel

Assemblersyntax:

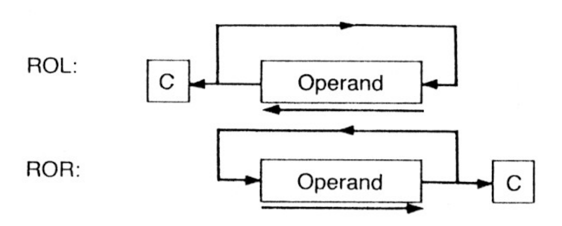
ROL Dx,Dy ; ROR Dx,Dy
oder
ROL #<Daten>,Dn ; ROR #<Daten>,Dn
oder
ROL <EA> ; ROR <EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Die Bits eines Operanden werden entsprechend der Richtung L oder R ringverschoben. Die Anzahl der Stellen, mit der die Ringverschiebung durchgeführt wird, kann auf zwei unterschiedliche Arten angegeben werden:

- durch einen unmittelbar im Befehl enthaltenen Wert;
- über ein Register.

Ein Speicherfeld kann immer nur um ein Bit maximal verschoben werden. Die zugelassene Operandenlänge ist dabei ein Wort.



Befehlsformat: Ringverschiebung eines Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Anzahl/ Register			dr	Operations- modus		i/r	1	1	Register		

Bezeichnungen:

Anzahl/Register – gibt die Stellenanzahl für die Ringverschiebung oder das Register, das den Verschiebungsfaktor enthält, an.

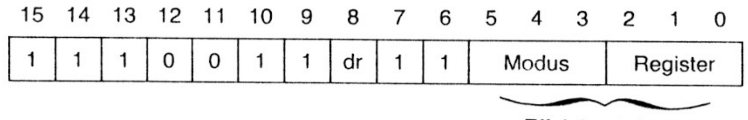
i/r – i/r=0 – Verschiebungsfaktor ist in den Bits 9 bis 11 des Befehls enthalten. Dabei entsprechen die Werte 0, 1,...,7 einer Verschiebung von 8, 1,...,7.
i/r=1 – Verschiebungsfaktor (modulo 64) ist im angegebenen Datenregister enthalten.

dr – gibt die Richtung der Ringverschiebung an.
dr=0 – rechts
dr=1 – links

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort

Register – definiert die Nummer des Datenregisters, in dem der Verschiebungsfaktor steht.

Befehlsformat: Ringverschiebung eines Speicherfeldes



Bezeichnungen:

dr – gibt die Richtung der Ringverschiebung an.
dr=0 – rechts
dr=1 – links

Effektive Adresse – gibt den Operanden an, in dem der Verschiebungsfaktor enthalten ist.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	B

N wird auf 1 gesetzt, wenn das höchstwertige Bit des Ergebnisses 1 ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird durch das letzte aus dem Operanden herausgeschobene Bit bestimmt. Es wird auf 0 gesetzt, wenn keine Verschiebung stattgefunden hat.
X nicht beeinflußt.

ROXL, ROXR ✳ *Rotate Left, Right with Extend Ringverschiebung links, rechts mit Erweiterungsbit*

Operation: (Ziel)ringverschoben um<Stellenzahl>→Ziel

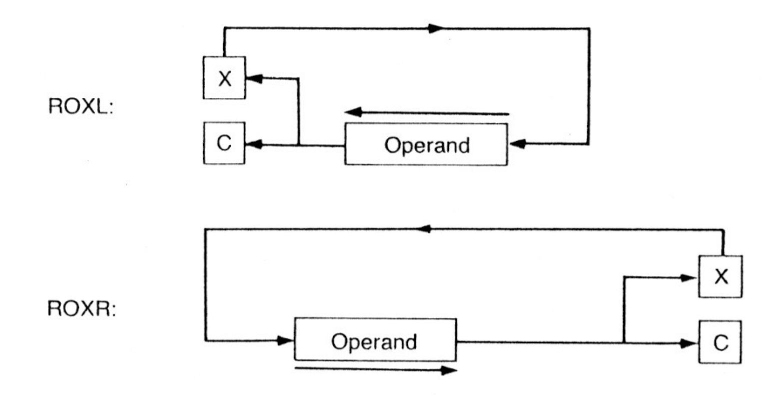
Assemblersyntax: ROXL Dx,Dy ; ROXR Dx,Dy
oder
ROXL #<Daten>,Dn; ROXR #<Daten>,Dn
oder
ROXL <EA> ; ROXR <EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Die Bits eines Operanden einschießlich eines Erweiterungsbits werden entsprechend der Richtung L oder R ringverschoben. Die Anzahl der Stellen, mit der die Ringverschiebung durchgeführt wird, kann auf zwei unterschiedliche Arten angegeben werden:

– durch einen unmittelbar im Befehl enthaltenen Wert;
– über ein Register.

Ein Speicherfeld kann immer nur um ein Bit maximal verschoben werden. Die zugelassene Operandenlänge ist dabei ein Wort.



Befehlsformat: Ringverschiebung eines Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Anzahl/ Register			dr	Operations- modus		i/r	1	0	Register		

Bezeichnungen:

Anzahl/Register – gibt die Stellenanzahl für die Ringverschiebung oder das Register, das den Verschiebungsfaktor enthält, an.

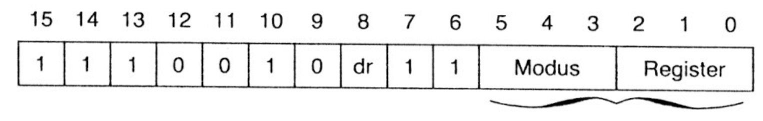
i/r – i/r=0 – Verschiebungsfaktor ist in den Bits 9 bis 11 des Befehls enthalten. Dabei entsprechen die Werte 0, 1,...,7 einer Verschiebung von 8, 1,...,7.
i/r=1 – Verschiebungsfaktor (modulo 64) ist im angegebenen Datenregister enthalten.

dr – gibt die Richtung der Ringverschiebung an.
dr=0 – rechts
dr=1 – links

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort

Register – definiert die Nummer des Datenregisters, in dem der Verschiebungsfaktor steht.

Befehlsformat: Ringverschiebung eines Speicherfeldes



Bezeichnungen:

dr – gibt die Richtung der Ringverschiebung an.

dr=0 – rechts
dr=1 – links
Effektive Adresse – gibt den Operanden an, in dem der Verschiebungsfaktor enthalten ist.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
B	B	B	0	B

N wird auf 1 gesetzt, wenn das höchstwertige Bit des Ergebnisses 1 ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird durch das letzte aus dem Operanden herausgeschobene Bit bestimmt. Es wird auf 0 gesetzt, wenn keine Verschiebung stattgefunden hat.
X wird durch das letzte aus dem Operanden herausgeschobene Bit bestimmt. Es bleibt unbeeinflußt, wenn keine Verschiebung stattgefunden hat.

RTE

Return from Exception
Rückkehr aus Ausnahmezustand

Operation: Wenn der Prozessor im Supervisor-Modus ist, dann:

(SP)+→SR
(SP)+→PC

Sonst Ausnahmezustand „Privilegverletzung".

Assemblersyntax: RTE

Operandenlänge: –

Beschreibung: Die Inhalte des Statusregisters SR und des Programmzählers PC werden durch Zurückholen vom Systemstapelspeicher wiederhergestellt. Die z. Z. aktuellen Inhalte des SR und des PC werden nicht gerettet. Wenn der Prozessor nicht im Supervisor-Modus ist, geht er in den Ausnahmezustand „Privilegverletzung" mit der Vektornummer 8 über.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

Bedingungscodes: Die Bedingungscodes werden entsprechend dem vom Stapel geholten Wort gesetzt.

RTR

Return and Restore Condition Codes
Rückkehr und Wiederherstellung der Bedingungscodes

Operation: (SP)+ → CCR
(SP)+ → PC

Assemblersyntax: RTR

Operandenlänge: –

Beschreibung: Die Inhalte des Bedingungscoderegisters CCR und des Programmzählers PC werden vom Stapel geholt. Die z. Z. aktuellen Inhalte des CCR und des PC werden nicht gerettet. Das Supervisor-Byte des Statusregisters bleibt unbeeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

Bedingungscodes: Die Bedingungscodes werden entsprechend dem vom Stapel geholten Wort gesetzt.

RTS

Return from Subroutine
Rückkehr aus dem Unterprogramm

Operation: (SP)+→PC

Assemblersyntax: RTS

Operandenlänge: –

Beschreibung: Der Inhalt des Programmzählers PC wird vom Stapel geholt. Der z. Z. aktuelle Inhalt des PC wird nicht gerettet.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

Bedingungscodes: nicht beeinflußt.

SBCD * *Subtract Decimal with Extend* / *Dezimale Subtraktion mit Erweiterungsbit*

Operation: $(\text{Ziel})_{10} - (\text{Quelle})_{10} - X \rightarrow \text{Ziel}$

Assemblersyntax: SBCD Dy,Dx
oder
SBCD −(Ay),−(Ax)

Operandenlänge: Byte

Beschreibung: Subtraktion des Quelloperanden und des Erweiterungsbits vom Zieloperanden, wobei der Zieloperand mit dem Ergebnis überschrieben wird. Die Subtraktion wird arithmetisch für binär codierte Dezimalzahlen durchgeführt. Die Operanden sind entweder bei direkter Adressierung in Datenregistern oder beim Predekrementierungs-Modus in Speicherfeldern enthalten.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0		Register Rx		1	0	0	0	0	R/M		Register Ry	

Bezeichnungen:

Register Rx – gibt das Zielregister an.
R/M=0 – Rx ist ein Datenregister.
R/M=1 – Rx ist ein Adreßregister, das für die Predekrementierung verwendet wird.

R/M=0 – Operand ist in einem Datenregister enthalten.

R/M=1 – Operand ist in einem Speicherfeld enthalten.

Register Ry – gibt das Quellregister an.
R/M=0 – Ry ist ein Datenregister.

R/M=1 – Ry ist ein Adreßregister, das für die Predekrementierung verwendet wird.

Bedingungscodes:

X	N	Z	V	C
B	U	B	U	B

N undefiniert.
Z wird auf 0 gesetzt, wenn das Ergebnis nicht 0 ist, sonst unverändert.
V undefiniert.
C wird auf 1 gesetzt, wenn ein dezimaler Übertrag anfällt, sonst auf 0.
X wie C.

Bei Operationen mit mehrfacher Genauigkeit ist es ratsam, vor der Ausführung Z mit 1 zu initialisieren (siehe auch Programm 7 im Kapitel „Anwenderprogramme“).

Scc

Set according to Condition
Bedingtes Setzen eines Bytes

Operation: Wenn der Bedingungsausdruck wahr ist,

dann \$FF→Ziel
sonst \$00→Ziel

Assemblersyntax: Scc <EA>

Operandenlänge: Byte

Beschreibung: Der angegebene Bedingungsausdruck wird geprüft. Wenn der Test „Wahr" ergibt, wird das durch die effektive Adresse angegebene Byte auf \$FF gesetzt (Operand ist wahr). Ansonsten wird es auf 0 gesetzt (Operand ist falsch). Mit einem auf Scc folgenden NEG-Befehl ist es möglich, dem Byte sowohl den Wert \$00 als auch den Wert \$01 zuzuweisen. Die für cc zulässigen Bedingungen können der Tabelle für den Befehl DBcc entnommen werden.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Bedingung				1	1	Modus			Register		
										Effektive Adresse					

Bezeichnungen:

Bedingung – ist eine der 16 Bedingungen aus der Tabelle für den Befehl DBcc.

Effektive Adresse – gibt die Zieladresse an, wohin das resultierende Byte geschrieben werden soll.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes: nicht beeinflußt.

STOP

Load Statusregister and Stop
Laden des Statusregisters und Stop

Operation: Wenn der Prozessor im Supervisor-Modus ist, dann:

<unmittelbare Daten>→SR

Sonst Ausnahmezustand „Privilegverletzung“.

Assemblersyntax: STOP <16-Bit-Daten>

Operandenlänge: –

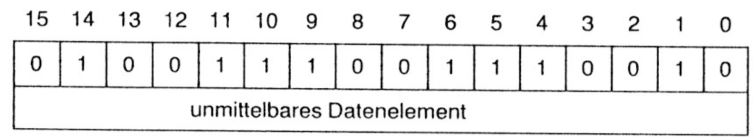
Beschreibung: Das unmittelbare Datenelement wird in das Statusregister SR gebracht. Der Programmzähler PC zeigt auf den nächsten auszuführenden Befehl, und der Prozessor stoppt. Die Verarbeitung wird fortgesetzt, wenn TRACE aktiviert wird, eine Unterbrechung oder ein RESET auftritt.

Die Ausnahme TRACE wird eingeleitet, wenn das Bit T=1 ist, während der STOP-Befehl ausgeführt wird.

Eine Unterbrechung kommt zum Zuge, wenn ihre Prioritätsebene höher als die derzeitige ist.

Ein externes RESET ruft immer die Ausnahmeverarbeitung RESET hervor. Wenn das dem Statusbit S entsprechende Bit des unmittelbaren Datenelements 0 ist, wird der Ausnahmezustand „Privilegverletzung“ erzeugt.

Befehlsformat:



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0
unmittelbares Datenelement															

Bedingungscodes: Sie werden entsprechend dem unmittelbaren Datenelement gesetzt.

SUB

Subtract Binary
Binäre Subtraktion

Operation: (Ziel)−(Quelle)→Ziel

Assemblersyntax: SUB <EA>,Dn
oder
SUB Dn,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Binäre Subtraktion des Quelloperanden vom Zieloperanden und Ersetzen des Zieloperanden durch das Ergebnis.

Befehlsformat:

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	0	1	Register	Operationsmodus	Modus	Register
						Effektive Adresse	

Bezeichnungen:

Register – gibt eins der 8 Datenregister an.

Operationsmodus

Byte	Wort	Doppelwort	Operation
000	001	010	(<Dn>)−(<EA>)→<Dn>
100	101	110	(<EA>)−(<Dn>)→<EA>

Effektive Adresse – bestimmt den Quelloperanden.

Dabei sind die folgenden Adressierungsarten zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	001	Nummer des Registers	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
– (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

* Für Byteoperationen ist die Adressierungsart „Adreßregister direkt" nicht erlaubt

Effektive Adresse – bestimmt den Zieloperanden.

Dabei sind die folgenden Adressierungsarten zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	–	–	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird bei Überlauf auf 1 gesetzt, sonst auf 0.
C wird auf 1 gesetzt, wenn ein dezimaler Übertrag angefallen ist, sonst auf 0.
X wie C.

SUBA *Subtract Address Subtraktion der Adresse*

Operation: (Ziel) – (Quelle) → Ziel

Assemblersyntax: SUBA <EA>,An

Operandenlänge: Wort, Doppelwort

Beschreibung: Subtraktion des Quelloperanden von dem Zieloperanden und Ersetzen des Zieloperanden durch das Ergebnis. Ein Quelloperand mit Wortlänge wird vor der Befehlsausführung vorzeichenbehaftet auf 32 Bits erweitert.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Register			Operationsmodus			Modus			Register		
										Effektive Adresse					

Bezeichnungen:

Register – gibt eins der 8 Adreßregister an. Sie werden stets als Ziel verwendet.

Operationsmodus – 011 – Wort. Der Operand wird auf 32 Bits erweitert, und die Operation wird mit einem 32-Bit-Adreßregister durchgeführt.
111 – Doppelwort

Effektive Adresse – bestimmt den Quelloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	001	Nummer des Registers	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	111	010
− (An)	100	Nummer des Registers	d(PC, Xi)	111	011
d(An)	101	Nummer des Registers	unmittelbar	111	100

Bedingungscodes: nicht beeinflußt.

SUBI

Subtract Immediate
unmittelbare Subtraktion

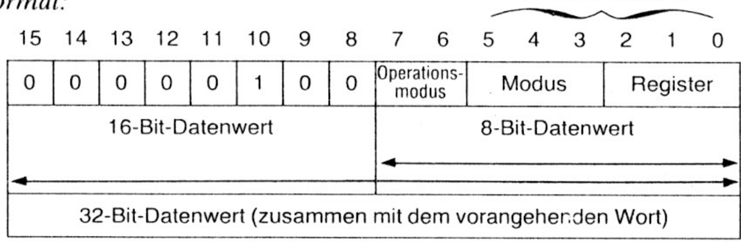
Operation: (Ziel) – <unmittelbare Daten> → Ziel

Assemblersyntax: SUBI #<Daten>,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Subtraktion des unmittelbaren Datenelementes von dem Zieloperanden und Ersetzen des Zieloperanden durch das Ergebnis.

Befehlsformat:



Bezeichnungen:

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort
Effektive Adresse – bestimmt den Zieloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird bei Überlauf auf 1 gesetzt, sonst auf 0.
C wird auf 1 gesetzt, wenn ein dezimaler Übertrag angefallen ist, sonst auf 0.
X wie C.

SUBQ

Subtract Quick
schnelle Subtraktion

Operation: (Ziel)−<unmittelbare Daten>→Ziel

Assemblersyntax: SUBI #<Daten>,<EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Subtraktion des unmittelbaren Datenelementes von dem Zieloperanden. Die Daten können die Werte 1 bis 8 annehmen. Die Wort- und Doppelwortoperationen können auch mit Adreßregistern durchgeführt werden, aber in diesem Fall werden die Bedingungscodes nicht gesetzt. Die Quelloperanden bei Wortoperationen werden vor der Befehlsausführung auf 32 Bits vorzeichenbehaftet erweitert.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Datenwert			1	Operations-modus		Modus			Register		
										Effektive Adresse					

Bezeichnungen:

Datenwert – ist unmittelbar in 3 Bits codiert.
0 – Subtraktion von 8.
1,...,7 – Subtraktion von 1 bis 7.

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort

Effektive Adresse – bestimmt den Zieloperanden.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	001	Nummer des Registers	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

* Nur Wort und Doppelwort

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Ergebnis 0 ist, sonst auf 0.
V wird bei Überlauf auf 1 gesetzt, sonst auf 0.
C wird auf 1 gesetzt, wenn ein negativer Übertrag angefallen ist (Zieloperand ist kleiner als das Datenwort), sonst auf 0.
X wie C.

SUBX *Subtract with Extend* *Subtraktion mit Erweiterungsbit*

Operation: (Ziel)−(Quelle)−X→Ziel

Assemblersyntax: SUBX Dy, Dx
oder
SUBX −(Ay),−(Ax)

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Subtraktion des Quelloperanden und des Erweiterungsbit von dem Zieloperanden und Überschreiben des Zieloperanden mit dem Ergebnis. Die Operanden können auf zwei verschiedene Arten angegeben werden: in Datenregistern und im Direkt-Modus mit Predekrementierung.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Register Rx			1	Operationsmodus		0	0	R/M	Register Ry		

Bezeichnungen:

Register Rx – gibt das Zielregister an.
R/M=0 – Datenregister.
R/M=1 – Adreßregister mit Predekrementierung.

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort

Register Ry – gibt das Quellregister an.
R/M=0 – Datenregister.
R/M=1 – Adreßregister mit Predekrementierung.

Bedingungscodes:

X	N	Z	V	C
B	B	B	B	B

N wird auf 1 gesetzt, wenn das Ergebnis negativ ist, sonst auf 0.
Z wird auf 0 gesetzt, wenn das Ergebnis nicht 0 ist, sonst bleibt es unverändert.
V wird bei Überlauf auf 1 gesetzt, sonst auf 0.
C wird auf 1 gesetzt, wenn ein negativer Übertrag angefallen ist (Zieloperand ist kleiner als das Datenwort), sonst auf 0.
X wie C.

Es ist notwendig, vor mehrfach genauen Operationen das Bit Z mit 1 zu initialisieren (siehe auch Programm 7 im Kapitel „Anwenderprogramm").

SWAP *Swap Register Halves* *Vertauschung von Registerhälften*

Operation: Registerbits(31–16)↔Registerbits(15–0)

Assemblersyntax: SWAP Dn

Operandenlänge: Wort

Beschreibung: Vertauschen der 16 unteren Bits mit den 16 oberen Bits eines Registers.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	Register		

Bezeichnungen:

Register – gibt das Datenregister an, dessen Hälften miteinander vertauscht werden sollen.

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das höchstwertige Bit (Bit 31) des Ergebnisses 1 ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das 32-Bit-Ergebnis 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird immer auf 0 gesetzt.
X nicht beeinflußt.

TAS * *Test and Set Operand* *Test und Setzen von Operanden*

Operation: (Getestetes Ziel)→Setzen CC
1→Bit 7 des Ziels

Assemblersyntax: TAS <EA>

Operandenlänge: Byte

Beschreibung: Das durch die effektive Adresse angegebene Byte wird getestet, und gemäß dem Testergebnis werden die Bedingungscodes N und Z gesetzt.

Das Bit 7 des Bytes wird auf 1 gesetzt. Dieser Befehl löst einen nicht unterbrechbaren Lese-/Änderungs-/Schreibzyklus aus, der eine Synchronisation mehrerer Prozessoren ermöglicht.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
0	1	0	0	1	0	1	0	1	1	Modus	Register

Effektive Adresse

Bezeichnungen:

Effektive Adresse – gibt die Adresse des zu testenden Operanden an.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn das höchstwertige Bit des Bytes 1 ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn das Byte 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird immer auf 0 gesetzt.
X nicht beeinflußt.

TRAP *Trap* *Übergang in den Ausnahmezustand*

Operation: PC→–(SSP)
SR→–(SSP)
(Vektor)→PC

Assemblersyntax: TRAP #<Vektor>

Operandenlänge: –

Beschreibung: Der Prozessor startet eine Ausnahmeprozedur. Die zugehörige Vektornummer wird durch 4 Bits innerhalb des Befehls angegeben. Es können also 16 verschiedene Vektoren für den TRAP-Befehl verwendet werden.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	Vektor			

Bezeichnungen:

Vektor – gibt dem Prozessor an, an welcher Adresse sich der in den PC zu ladende Wert befindet.

Bedingungscodes: nicht beeinflußt.

TRAPV *Trap and Overflow* *Übergang in den Ausnahmezustand bei Überlauf*

Operation: Wenn V=1, dann Übergang in den Ausnahmezustand TRAPV.

Assemblersyntax: TRAPV

Operandenlänge: –

Beschreibung: Der Prozessor startet eine Ausnahmeprozedur für Überlauf, wenn das entsprechende Zustandsbit V=1 ist. Für die erzeugte Ausnahmebedingung ist die Vektornummer 7 zuständig. Wenn die Bedingung nicht erfüllt ist, wird in der normalen Befehlssequenz fortgefahren.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

Bedingungscodes: nicht beeinflußt.

TST

Test an Operand
Test eines Operanden

Operation: (Ziel)getestet→Setzen von CC

Assemblersyntax: TST <EA>

Operandenlänge: Byte, Wort, Doppelwort

Beschreibung: Der Operand wird mit 0 verglichen. Die Bedingungscodes werden entsprechend dem Ergebnis gesetzt. Dabei erfolgt keine Wertänderung des Operanden.

Befehlsformat:

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
0	1	0	0	1	0	1	0	Operationsmodus	Modus	Register

Effektive Adresse (Modus, Register)

Bezeichnungen:

Operationsmodus – 00 – Byte
01 – Wort
10 – Doppelwort
Effektive Adresse – gibt die Länge des zu testenden Operanden an.

Die folgenden Adressierungsarten sind zulässig:

Adressierungsart	Modus	Register	Adressierungsart	Modus	Register
Dn	000	Nummer des Registers	d(An, Xi)	110	Nummer des Registers
An	–	–	Abs.W	111	000
(An)	010	Nummer des Registers	Abs.L	111	001
(An) +	011	Nummer des Registers	d(PC)	–	–
– (An)	100	Nummer des Registers	d(PC, Xi)	–	–
d(An)	101	Nummer des Registers	unmittelbar	–	–

Bedingungscodes:

X	N	Z	V	C
–	B	B	0	0

N wird auf 1 gesetzt, wenn der Operand negativ ist, sonst auf 0.
Z wird auf 1 gesetzt, wenn der Operand 0 ist, sonst auf 0.
V wird immer auf 0 gesetzt.
C wird immer auf 0 gesetzt.
X nicht beeinflußt.

UNLK *Unlink* *Lösen*

Operation: An→SP
(SP)+→An

Assemblersyntax: UNLK An

Operandenlänge: –

Beschreibung: Der Inhalt des angegebenen Adreßregisters wird in den Stapelzeiger geladen. Anschließend wird das oberste Langwort des Stapels in das Adreßregister geladen (bei einer Stapelverwaltung von hohen zu niedrigen Adressen).

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	Register		

Bezeichnungen:

Register – gibt das Adreßregister an.

Bedingungscodes: nicht beeinflußt.

DETAILLIERTE BETRACHTUNG EINZELNER BEFEHLE

DER BEFEHL Bcc

Assemblersyntax: Bcc <Label>

Beschreibung:

Wenn die Bedingung erfüllt ist, wird die Programmausführung an der Adresse (PC)+Adreßdistanzwert fortgesetzt. Die Adresse wird dem Befehl selbst mitgegeben. Die Verschiebung wird ebenfalls unmittelbar im Befehlswort angegeben oder, wenn diese Bits 0 enthalten sind, ist sie in den 16 Bits des Erweiterungswortes enthalten.

Der Distanzwert wird automatisch auf 8 Bits gesetzt, wenn es sich bei dem Befehl um einen kurzen Verzweigungsbefehl handelt:

Bcc.S xxx

Die Liste der möglichen Bedingungen ist in der Abb. 5.2 aufgeführt.

0100	CC	Carry Clear	kein Übertrag	C = 0
0101	CS	Carry Set	Übertrag	C = 1
0111	EQ	EQual	gleich	Z = 1
0110	NE	Not Equal	ungleich	Z = 0
1100	* GE	Greater or Equal	größer oder gleich	N + V = 0
1110	* GT	Greater Than	größer	Z + (N + V) = 0
0010	HI	HIgh	größer (ohne Vorzeichen)	C + Z = 0
1111	* LE	Less or Equal	kleiner oder gleich	Z + (N + V) = 1
0011	LS	Less or Same	kleiner oder gleich (ohne Vorzeichen)	C + Z = 1
1101	* LT	Less Than	kleiner	N + V = 1
1011	MI	MInus	negativ	N = 1
1010	PL	Plus	positiv	N = 0
1000	* VC	oVerflow Clear	kein Überlauf	V = 0
1001	* VS	oVerflow Set	Überlauf	V = 1
0000	• T	True	wahr	1
0001	• F	False	falsch	0

• Bei Bcc nicht verwendet
* Benutzt im Zweierkomplement

Abb. 5.2: Mögliche Bedingungen für Bcc

Beispiele:

```
BEQ SCHLEIFE
BNE.S AUSGANG
```

Wenn das Programm mit einem Assembler für den 68000, wie er auf dem MOTOROLA-System EXORMACS vorhanden ist, umgewandelt wird und wenn die Erweiterung eines Befehls Bcc nicht angegeben wurde, werden folgende Maßnahmen automatisch getroffen: Falls das Label vor dem Aufruf definiert wird, wird der Befehl Bcc als Bcc.S assembliert, sofern dies möglich ist. Wurde das Label erst nach dem Aufruf definiert, wird der Befehl wie Bcc.L behandelt. Dies wird durch das folgende Programm noch einmal verdeutlicht.

```
                00002000          ORG    $2000
00002000        4E71        L1    NOP
00002002        67FC              BEQ.S  L1
00002004        66FA              BNE    L1
00002006        6704              BEQ.S  L2
00002008        66000002          BNE    L2
WARNING 550
0000200C        4E71        L2    NOP
                                  END
TOTAL ERRORS          0
TOTAL WARNINGS        1
```

Bei diesem Beispiel fällt auf, daß der erste BNE-Befehl eine kurze Verzweigung ist, weil L1 schon zu Beginn des Programms definiert wurde. Der zweite BNE ist jedoch eine lange Verzweigung, denn L2 wird erst nach diesem Befehl definiert.

Die Warnung weist darauf hin, daß BNE eigentlich als kurzer Befehl hätte angegeben werden müssen.

```
                00002000          ORG    $2000
00002000        4E71        L1    NOP
00002002        4E71        L2    NOP
00002004        67FA              BEQ.S  L1
00002006        66F8              BNE    L1
00002008        67F8              BEQ.S  L2
0000200A        66F6              BNE    L2
END
TOTAL ERRORS          0
TOTAL WARNINGS        0
```

Die zwei BNE sind kurze Verzweigungsbefehle, da L1 und L2 schon zu Beginn des Programms definiert wurden und der Distanzwert nicht größer als 8 Bits ist.

```
                00002000          ORG    $2000
00002000        4E71        L1    NOP
00002002        4E71        L2    NOP
00002004        67FA              BEQ.S  L1
00002006        6600FFF8          BNE.L  L1
WARNING 550
0000200A        67F6              BEQ.S  L2
0000200C        6600FFF4          BNE.L  L2
WARNING 550
                                  END
TOTAL ERRORS           0
TOTAL WARNINGS         2
```

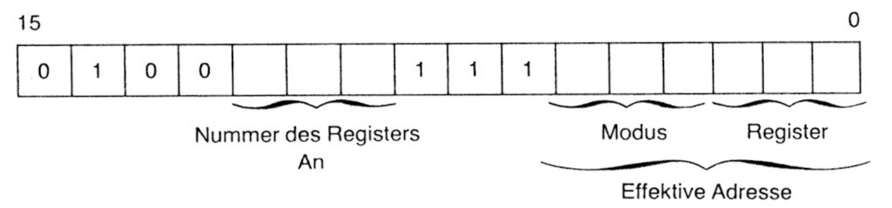
DER BEFEHL LEA

Assemblersyntax: LEA <EA>,An

Beschreibung:

Die effektive Adresse wird in das angegebene Adreßregister geladen. Die 32 Bits dieses Adreßregisters werden durch den Befehl beeinflußt.

Befehlsformat:



Beispiel:

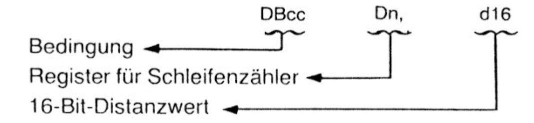
```
                00001000          ORG    $1000
00001000        43F83000          LEA    $3000, A1
                                  END
TOTAL ERRORS           0
TOTAL WARNINGS         0
```

Programmlauf

D0-D7	26534354	30303033	46495820	00000000
	20202020	20202020	00000000	00000061
A0-A7	45533120	00003000	00000000	30303033
	000012BD	010E0DEA	00000000	00000000

DER BEFEHL DBcc

Assemblersyntax: DBcc Dn,D16

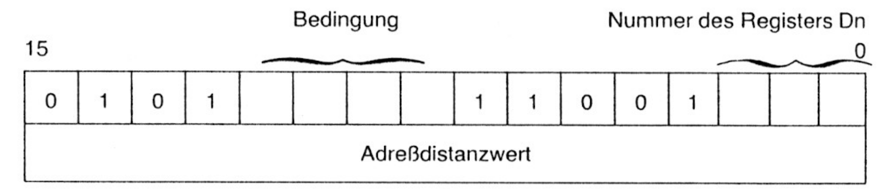


Beschreibung:

Der Prozessor testet, ob die Bedingung erfüllt ist. Falls die Bedingung wahr ist, wird als Befehl ein NOP ausgeführt, und die Ausführung wird mit dem nächstfolgenden Befehl fortgesetzt. Ist die Bedingung falsch, dann wird das angegebene Datenregister Dn um 1 vermindert.

- Ist das Ergebnis −1, wird dieser Befehl beendet, und es wird zum nächsten Befehl übergegangen.
- Ist das Ergebnis nicht −1, führt der Prozessor den Befehl aus, der sich an der Adresse befindet, die durch den PC und den auf 32 Bit erweiterten Distanzwert festgelegt ist.

Befehlsformat:



Der prinzipielle Ablauf dieses Befehls wird durch das Flußdiagramm in der Abb. 5.3 dargestellt.

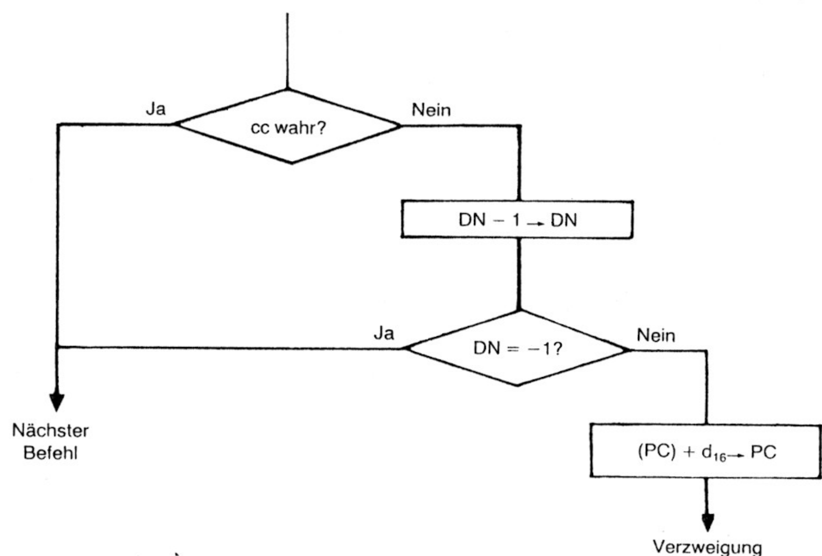


Abb. 5.3: Flußdiagramm des Befehls DBcc

Bemerkung: Dieser Befehl verändert die Bedingungscodes nicht. Falls die Bedingung F (False, Falsch) ist, akzeptieren die meisten Assembler den Mnemonik-Code DBRA anstelle von DBF.

Beispiel für die Verwendung des Befehls DBcc:

Stellen wir uns einmal vor, daß wir einen Speicherbereich M1 in einen anderen Speicherbereich M2 kopiert haben. Wir wollen nun gerne überprüfen, ob sich bei der Übertragung kein Fehler eingeschlichen hat. Sobald eine Abweichung im Duplikat entdeckt wird, wird automatisch eine entsprechende Verarbeitung durchgeführt. In diesem Beispiel liegt aber der Schwerpunkt unseres Interesses nicht auf dieser Folgeverarbeitung. Wir nehmen ferner an, daß der Speicherbereich 10 Worte enthält. Für M1 verwenden wir einen Adreßzeiger A1. Der entsprechende Zeiger für M2 heißt A2. Unseren Schleifenzähler nennen wir D0.

Die Initialisierungsphase des Programms lautet folgendermaßen:

```
LEA      $2000, A1     ;Laden von A1 mit $2000
LEA      $3000, A2     ;Laden von A2 mit $3000
MOVEQ    #9, D0        ;Laden von D0 mit 9, da D0 von 9 bis -1
                       ;dekrementiert wird, was 10 Durchläufen
                       ;entspricht.
```

Der Vergleich der beiden Speicherbereiche mit Postinkrementierung wird wie folgt realisiert:

SCHLEIFE CMPM (A1)+,(A2)+

Das zugehörige Flußdiagramm ist in Abb. 5.4 dargestellt.

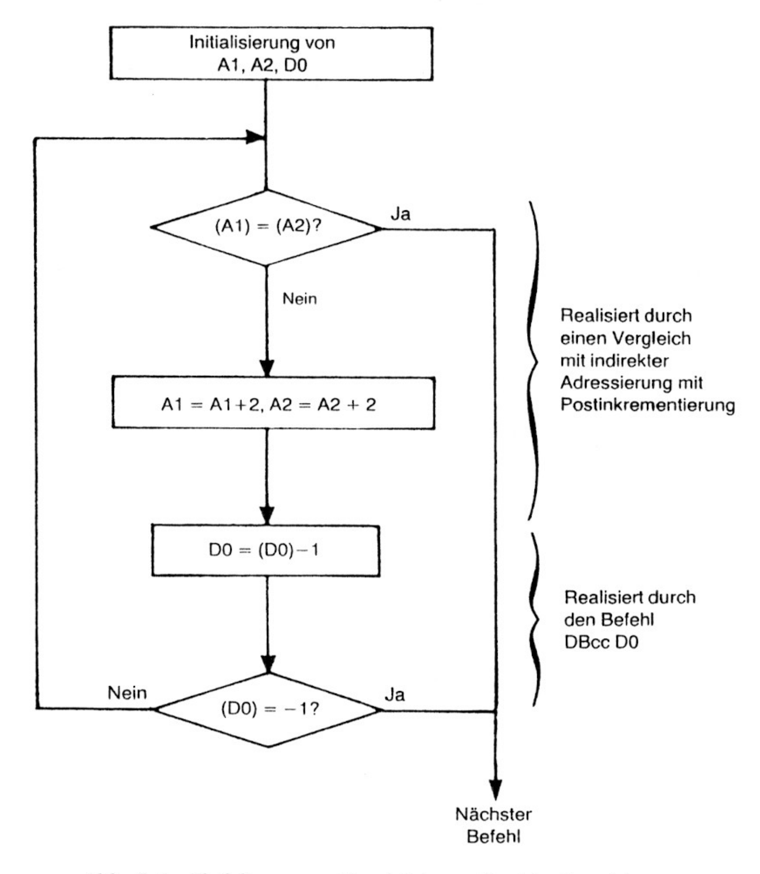


Abb. 5.4: Flußdiagramm Vergleich von Speicherbereichen

Der Befehl für den Test auf Schleifenende lautet:

DBNE D0,SCHLEIFE

Solange Gleichheit zwischen (A1) und (A2) besteht und D0 ungleich −1

ist, wird der Befehl CMPM weiter ausgeführt. Das vollständige Programm sieht also folgendermaßen aus:

```
         ORG        $1000
         LEA        $2000, A1
         LEA        $3000, A2
         MOVEQ      #9, D0
SCHLEIFE
         CMPM       (A1)+, (A2)+
         DBNE       D0, SCHLEIFE
         Folgebefehle
         .
         .
         .
         .
         .
         END
```

Die weiteren Befehle des Programms sind durch die Bemerkung „Folgebefehle“ symbolisiert. Dies könnte z. B. ein Test sein, der feststellt, ob D0 gleich −1 ist. Falls dies der Fall ist, wäre das Ende des Speicherbereichs erreicht. Während des Vergleichs der beiden Speicherbereiche auf Fehler kann nach einer entsprechenden Fehlererkennungsroutine zu einem zugehörigen Unterprogramm für die Fehlerbehandlung verzweigt werden.

In diesem Programm haben wir nun die Übereinstimmung von jeweils 10 Worten untersucht. Dies geschah unter der Kontrolle des Schleifenzählers D0, der mit 9 initialisiert wird, weil sich der DBNE-Befehl für die Schleifenüberwachung hinter dem Vergleichsbefehl CMPM befindet.

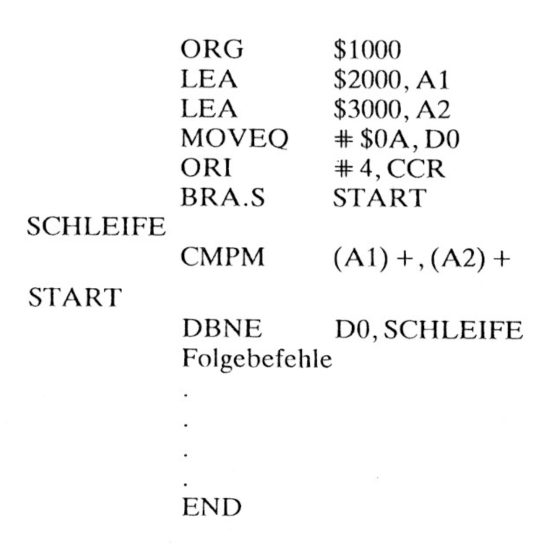
Wir nehmen nun an, daß sich der Befehl CMPM nach dem Test von DBNE befände.

– Während des ersten Schleifendurchlaufs ist der Zustand des Bedingungscodes Z noch nicht bekannt, und somit ist unklar, wie der Test DBNE ausgehen wird. Dennoch darf die Bedingung NE noch nicht als wahr bestätigt werden, bis der Befehl CMPM überhaupt einmal zur Ausführung gelangt. Es ist daher notwendig, Z mit 1 zu initialisieren. Dies wird durch folgenden Befehl erreicht:

  ORI #4,CCR

– Der Befehl CMPM wird nach dem Herunterzählen des Schleifenzählers D0 ausgeführt. D0 muß daher mit 10 initialisiert werden, wenn

man den Vergleich für 10 Worte durchführen möchte. Für das Programm ergibt sich also:



```
              ORG       $1000
              LEA       $2000, A1
              LEA       $3000, A2
              MOVEQ     #$0A, D0
              ORI       #4, CCR
              BRA.S     START
SCHLEIFE
              CMPM      (A1)+, (A2)+
START
              DBNE      D0, SCHLEIFE
              Folgebefehle
              .
              .
              .
              .
              END
```

Wir wollen nun ein anderes Beispiel betrachten, das die Bedeutung des Befehls DBF Dn,xxx (gleichbedeutend mit DBRA Dn,xxx) hervorhebt.

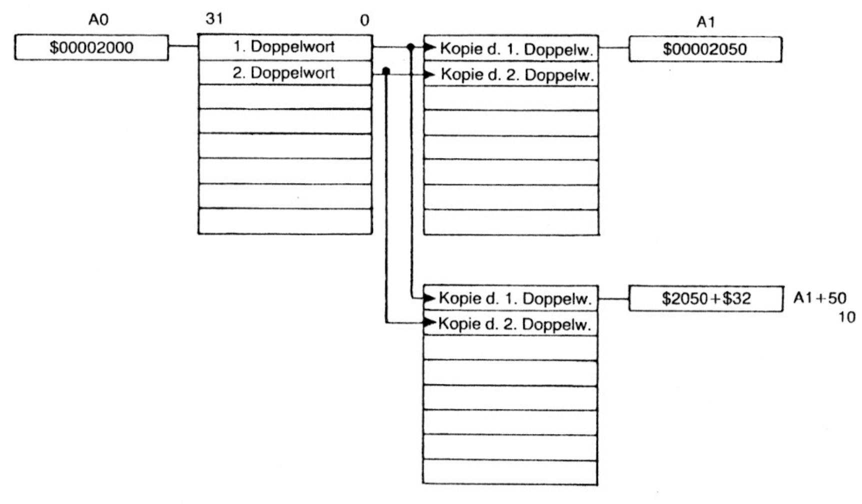


Abb. 5.5: Prinzip des Kopierens

Dieser bedingte Verzweigungsbefehl führt einen Test auf Ende einer Verarbeitung durch.

Solange Dn ungleich −1 ist, beginnt die Verarbeitung wieder von neuem. Andernfalls wird sie beendet.

Gleichzeitig soll uns das Beispiel die Rolle der indizierten Adressierung mit Adreßdistanzwert vorführen.

Inhaltlich geht es darum, 8 Doppelworte aus einer Tabelle mit Verschiebung zu kopieren (Abb. 5.5).

Das Register A0 markiert den Anfang der Ursprungstabelle.

Das Register A1 markiert den Anfang der ersten Zieltabelle. Die zweite befindet sich in einem relativen Abstand von 50 Bytes zu der ersten Zieltabelle.

Das Register D0 dient als Indexregister, und D1 zählt die Schleifendurchgänge.

Das Programm kann also folgendermaßen aussehen:

```
          00001000        ORG      $1000
00001000  41F82000        LEA      $2000, A0        ;Initialisierungen
00001000  43F82050        LEA      $2050, A1
00001008  7207            MOVEQ    #7, D1
0000100A  4280            CLR.L    D0               ;Nullsetzen des
                                                    ;Indexregisters
0000100C          SCHLEIFE
0000100C  23B000000000    MOVE.L   (A0, D0), (A1, D0)
                                                    ;Übertragung zur
                                                    ;ersten Tabelle
00001012  23B00000000A    MOVE.L   (A0, D0), $32 (A1, D0)
                                                    ;Übertragung zur
                                                    ;zweiten Tabelle
00001018  5880            ADDQ.L   #4, D0           ;Aktualisieren des
                                                    ;Indexregisters
0000101A  51C9FFF0        DBRA     D1, SCHLEIFE     ;Wenn D1 ungleich
                                                    ;−1 ist, erfolgt ein
                                                    ;weiterer Schleifen-
                                                    ;durchlauf
                          END
TOTAL ERRORS      0
TOTAL WARNINGS 0
```

DER BEFEHL EXT

Assemblersyntax: EXT.W Dn oder EXT.L Dn

Beschreibung:

Dieser Befehl erweitert das Vorzeichenbit des Datenregisters vom Byte zum Wort oder vom Wort zum Doppelwort. Wenn es sich also um eine Wortoperation handelt (.W), wird das Bit 7 des Registers Dn in die Bits 8 bis 15 kopiert. Ist es hingegen eine Doppelwortoperation (.L), so wird das Bit 15 von Dn in die Bits 16 bis 31 kopiert.

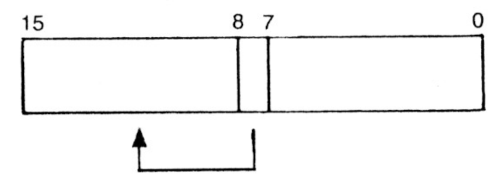


Abb. 5.6: Erweiterung vom Byte zum Wort: W

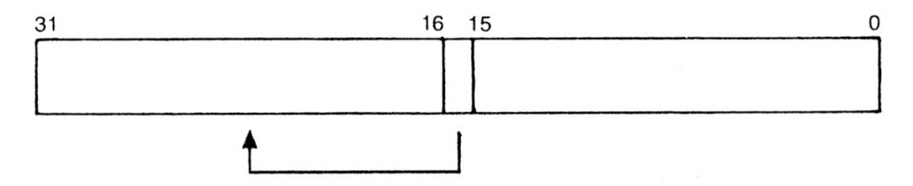


Abb. 5.7: Erweiterung vom Wort zum Doppelwort: L

DER BEFEHL SWAP

Assemblersyntax: SWAP Dn

Beschreibung:

Dieser Befehl bewirkt einen Austausch der oberen 16 Bits mit den unteren 16 Bits in ein und demselben Datenregister (Abb. 5.8).

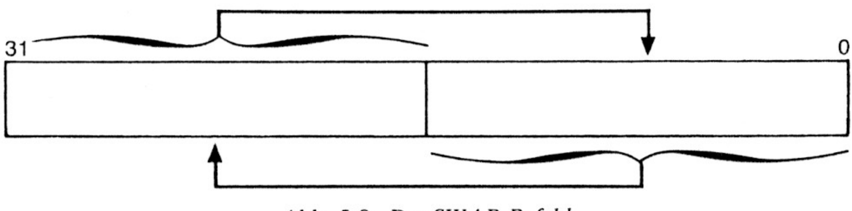
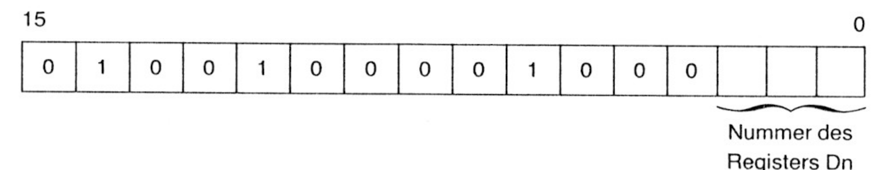


Abb. 5.8: Der SWAP-Befehl

Befehlsformat:



DIE BEFEHLE DIVS UND DIVU

Assemblersyntax: DIVS<E>,Dn, DIVU <EA>,Dn

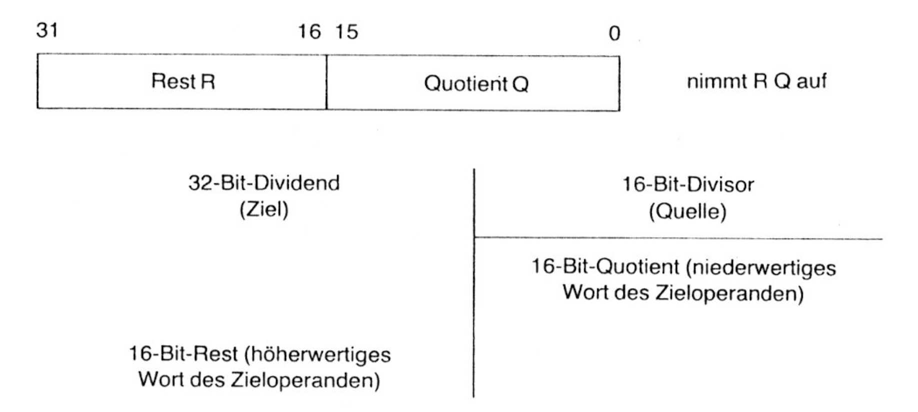
(Ziel)/(Quelle)→Ziel

Diese Befehle führen eine Division des Zieloperanden durch den Quelloperanden durch.

Der Zieloperand besteht aus 32 Bits (Datenregister), der Quelloperand aus 16.

Das Ergebnis, das mit 32 Bits in den Zieloperanden geschrieben wird, ist folgendermaßen aufgebaut:

- Der Quotient wird im niederwertigen Wort gespeichert.
- Der Rest wird im höherwertigen Wort abgelegt.

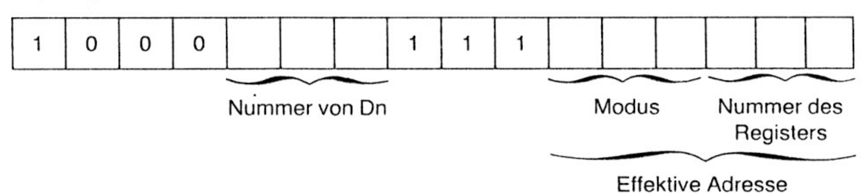


DIVS führt die Operation mit vorzeichenbehafteter Arithmetik durch, und das Vorzeichen des Restes ist das gleiche wie das des Dividenden (außer wenn es 0 ist).

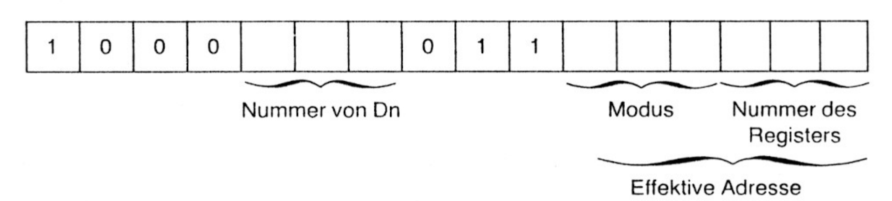
DIVU führt die Operation ohne Vorzeichen durch. Eine Division durch 0 erzeugt den Ausnahmezustand „Division durch 0", dessen Vektornummer 5 ist.

Wenn im Verlauf der Befehlsausführung ein Überlauf auftritt, wird der Bedingungscode V auf 1 gesetzt, und die Operanden bleiben unverändert (der Zieloperand enthält nach wie vor den Dividenden).

Befehlsformat von DIVS:



Befehlsformat von DIVU:



Beispiel: Division mit oder ohne Überlauf

Das Prinzip der Division mit Kapazitätsüberlauf beruht auf der Tatsache, daß man jede 32-Bit-Zahl YZ (Y stellt die 16 höherwertigen und Z die 16 niederwertigen Bits dar) auch folgendermaßen schreiben kann:

$$Y*2^{16}+Z$$

Die Division wird also für jede Stelle durchgeführt, indem die Reste der vorhergehenden Divisionen berücksichtigt werden.

Der Rechenalgorithmus ist in Abb. 5.9 dargestellt.

Der Dividend YZ sei im Datenregister D1 und der Divisor X in D0 gespeichert. Nach Beendigung des Programmablaufs wird D1 den Quotienten und D0 den Rest enthalten.

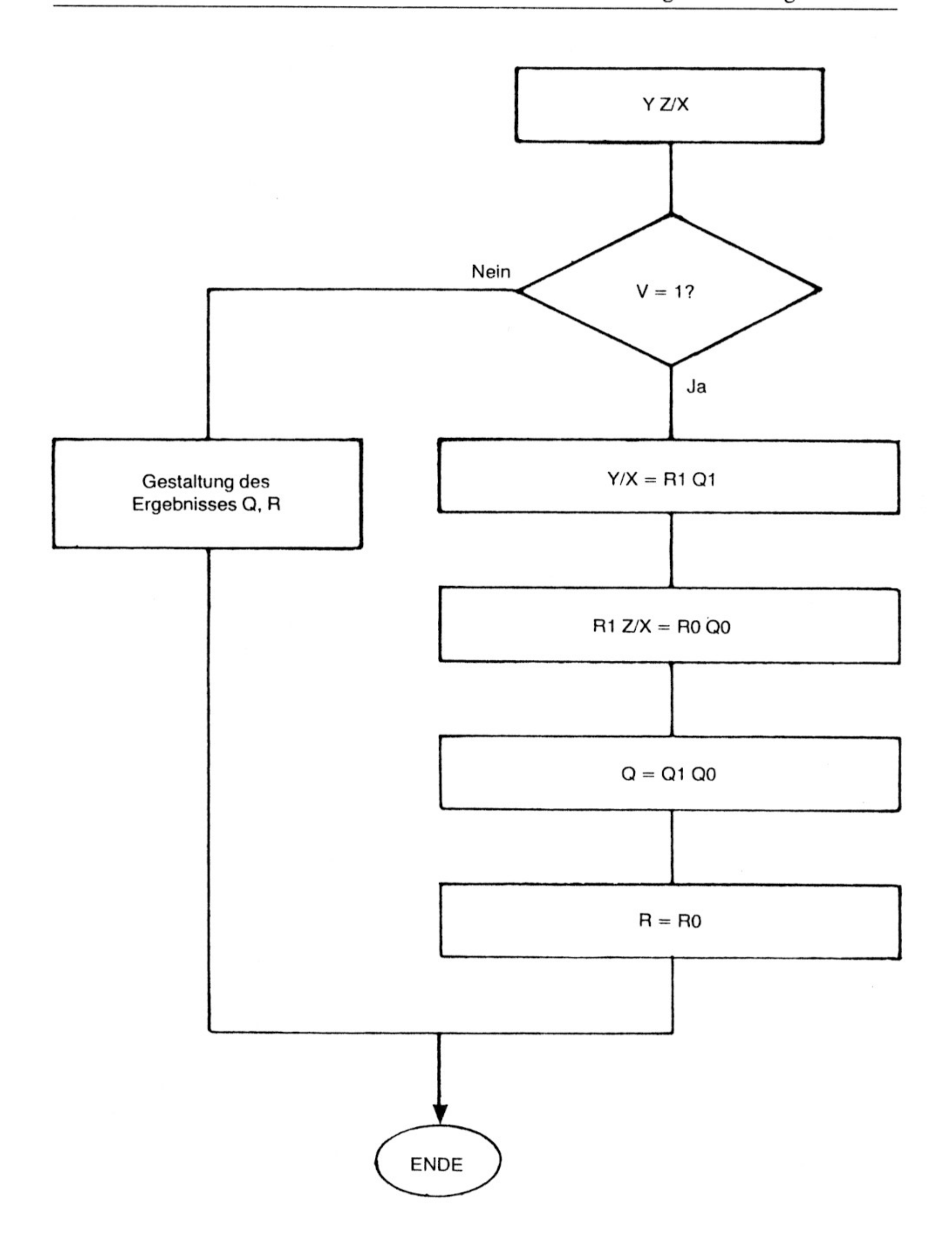


Abb. 5.9: Flußdiagramm der Division

Das Programm sieht nun folgendermaßen aus:

```
        CLR.L   D3      ;Nullsetzen des Registers D3, das als
                        ;temporärer Speicher benutzt wird
        DIVU    D0, D1  ;Y Z / X
        BVC     RESULT  ;Wenn V = 0 ist, wird zur Ergebnis-
                        ;gestaltung verzweigt
        MOVE.L  D1, D2  ;Retten von Y Z in D2
        CLR.W   D1      ;D1 wird von Y Z auf Y 0 gesetzt
        SWAP    D1      ;D1 wird von Y 0 auf 0 Y gesetzt
        DIVU    D0, D1  ;Y / Z = R1 Q1 in D1
        MOVE.W  D1, D3  ;Retten von R1 Q1 in D3
        MOVE.W  D2, D1  ;D1 wird von R1 Q1 auf R1 Z gesetzt
        DIVU    D0, D1  ;R1 Z / X = R0 Q0 in D1
RESULT
        MOVE.L  D1, D0  ;D0 = R Q oder besser R0 Q0
        SWAP    D1      ;D1 = Q R oder besser Q0 R0
        MOVE.W  D3, D1  ;D1 = Q 0 oder besser Q0 Q1
        SWAP    D1      ;D1 = 0 Q oder besser Q1 Q0
        CLR.W   D0      ;D0 = R 0 oder besser R0 0
        SWAP    D0      ;D0 = 0 R oder besser 0 R0
        END
```

DER BEFEHL Scc

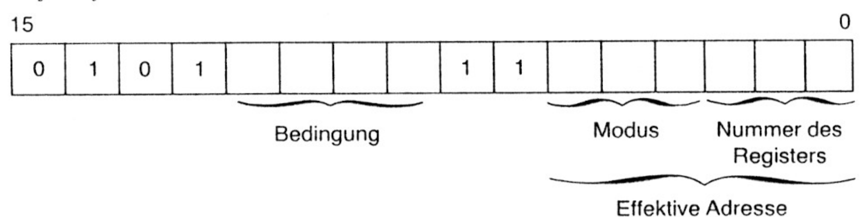
Assemblersyntax: Scc <EA>

Beschreibung:

Die Bedingung cc wird geprüft. Wenn sie erfüllt ist, wird das Byte, das durch die effektive Adresse bezeichnet ist, auf „Wahr" gesetzt, also auf den Wert $FF. Andernfalls wird es auf „Falsch" gesetzt, also auf 0.

Dieser Befehl wird natürlich gebraucht, wenn man Boolesche Variablen setzen will.

Befehlsformat:



Beispiel:

Setzen der Booleschen Variablen BGLEICH, wenn D0=D1 ist:

```
        CLR.B     BGLEICH ;Nullsetzen des Booleschen Wertes
                          ;BGLEICH
        CMP       D0, D1  ;Vergleich der beiden Register
        SEQ       BGLEICH ;Setzen von BGLEICH
        Folgebefehle

BGLEICH
        DS.B      1
        END
```

DIE BEFEHLE FÜR DEZIMALARITHMETIK: ABCD, SBCD UND NBCD

Diese drei Befehle realisieren arithmetische Operationen mit mehrfacher Genauigkeit, und zwar mit binärcodierten Dezimalzahlen.

ABCD

Assemblersyntax: ABCD Dy, Dx oder ABCD −(Ay),−(Ax)

Beschreibung:

Der Quelloperand wird zum Zieloperanden und dem Erweiterungsbit in binärcodierter Arithmetik addiert. Es handelt sich hierbei um eine Byteoperation.

$(\text{Quelle})_{10}+(\text{Ziel})_{10}+X \rightarrow \text{Ziel}$

Lediglich zwei Adressierungsarten sind zugelassen:

- direkte Datenregister-Adressierung
- indirekte Adressierung mit Predekrementierung

Besonders die zweite Adressierungsart ist in Zusammenhang mit diesem Befehlstyp interessant. Tatsächlich wird, wenn sich der Operand aus mehreren Bytes zusammensetzt, das höherwertige Byte an einer niedrigeren Adresse abgelegt als das niederwertige. Wenn also mit Predekrementierung gearbeitet wird, addiert man zuerst die unteren Bytes, dann die höheren usw. Das Ergebnis wird auf die gleiche Weise aufgebaut.

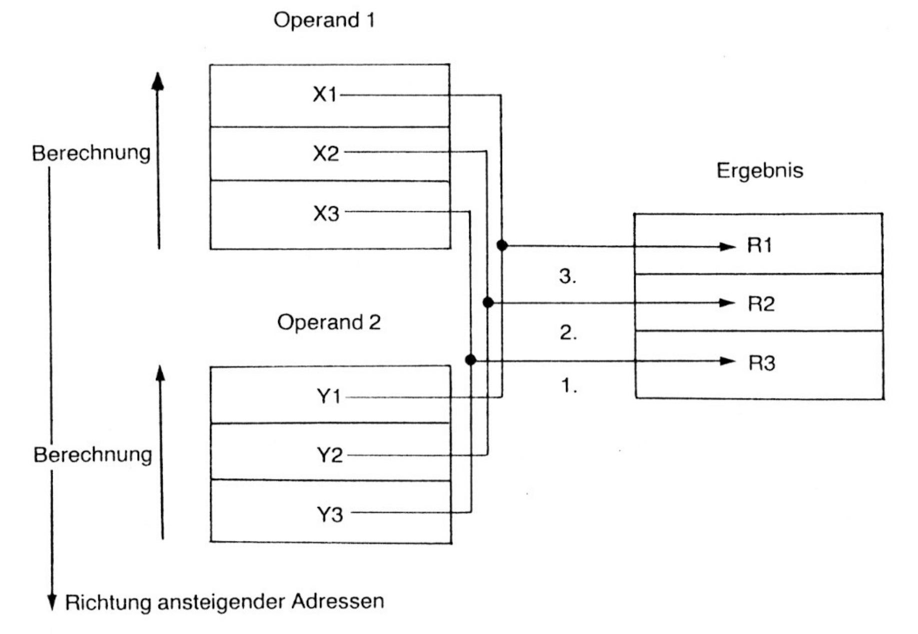
Sehen wir uns nun ein kleines Schema an: Dabei stellen X1, X2 und X3 die

Bytes dar, die die erste Zahl bilden, und Y1, Y2 und Y3 stellen die zweite Zahl dar, die jetzt addiert werden soll:

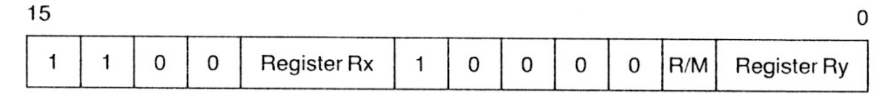
		X1	X2	X3	
(MSB)	+				(LSB)
		Y1	Y2	Y3	
		R1	R2	R3	

← Richtung der Berechnung

Vorgang im Speicher:



Befehlsformat:

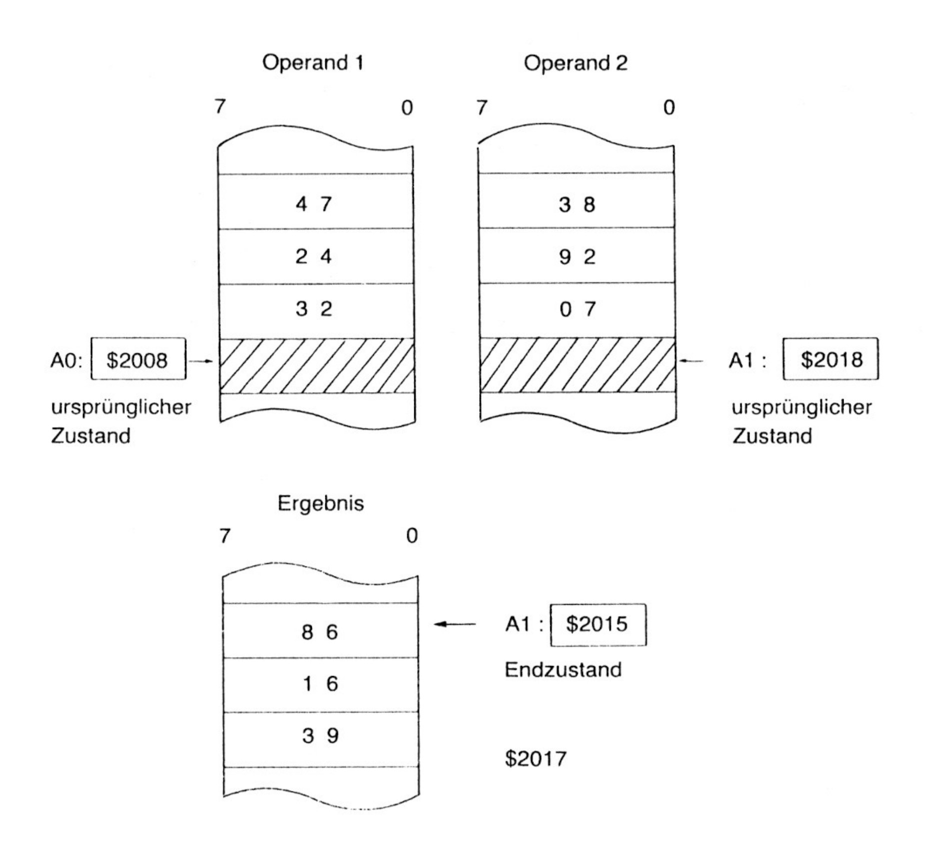


Rx: Nummer des Zielregisters
Ry: Nummer des Quellregisters

Wenn R/M = 0, dann Adressierungsart Datenregister direkt
Wenn R/M = 1, dann Adressierungsart indirekt mit Predekrementierung

Abb. 5.10: Format des Befehls ABCD

Beispiel:



Das Programm sieht folgendermaßen aus:

```
         00001000  ORG    $1000
00001000 41F82008  LEA    $2008, A0      ;Initialisierung
00001004 43F82018  LEA    $2018, A1      ;Initialisierung
00001008 7002      MOVEQ  $2, D0         ;Initialisierung des
                                         ;Schleifenzählers D0
```

```
0000100A            SCHLEIFE
0000100A   C308       ABCD    -(A0),-(A1)    ;BCD-Addition
0000100C   51C8FFF0   DBRA    D0,SCHLEIFE    ;Wenn D0 ungleich -1 ist,
                                             ;dann ein weiterer
                                             ;Schleifendurchlauf
                      END
TOTAL ERRORS  0
TOTAL ERRORS  0
```

SBCD

Assemblersyntax: SBCD Dy,Dx oder SBCD −(Ay),−(Ax)

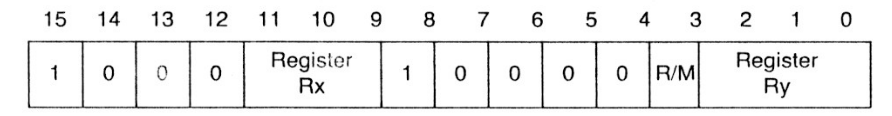
Beschreibung:

Der Quelloperand und das Erweiterungsbit werden vom Zieloperanden in binärcodierter Arithmetik subtrahiert.

Die erlaubten Adressierungsarten sind die gleichen wie im vorher beschriebenen Befehl ABCD.

$(\text{Ziel})_{10} - (\text{Quelle})_{10} - X \rightarrow \text{Ziel}$

Befehlsformat:



Register Rx: Bestimmt das Zielregister
Register Ry: Bestimmt das Quellregister

Wenn R/M = 0, dann Adressierungsart Datenregister direkt
Wenn R/M = 1, dann Adressierungsart indirekt mit Predekrementierung

Abb. 5.11: Format des Befehls SBCD

NBCD

Assemblersyntax: NBCD <EA>

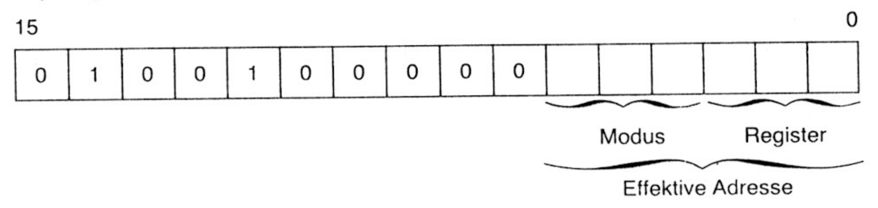
Beschreibung:

Der durch die effektive Adresse angesprochene Operand und das Erweiterungsbit werden in binärcodierter Arithmetik von 0 subtrahiert. Wenn

das Erweiterungsbit 0 ist, wird also das Zehnerkomplement des Operanden gebildet. Ist es 1, so handelt es sich um das Neunerkomplement.

0−(Ziel)10−X→Ziel

Befehlsformat:



DIE VERSCHIEBE- UND RINGVERSCHIEBUNGSBEFEHLE: ASR, ASL, LSR, LSL, ROXR, ROXL, ROR, ROL

Die Verschiebeoperationen werden durch die arithmetischen Befehle ASR (Rechtsverschiebung) und ASL (Linksverschiebung) und durch die logischen Befehle LSR (nach rechts) und LSL (nach links) ermöglicht.

Für die Ringverschiebung mit Hilfe des Erweiterungsbits X werden die Befehle ROXR (nach rechts), ROXL (nach links) zur Verfügung gestellt und ohne Erweiterungsbit die Befehle ROR (nach rechts) und ROL (nach links).

Die Arbeitsweise dieser verschiedenen Befehle wird mit Hilfe der schematischen Abb. 5.12 veranschaulicht.

Für alle diese Befehle wird der Verschiebungsfaktor entweder direkt im Befehl (unmittelbar) oder über ein angegebenes Datenregister spezifiziert.

Hinweis: Wenn die Verschiebung oder Ringverschiebung ein Datenregister betrifft, kann der Operand ein Byte, ein Wort oder ein Doppelwort sein. Im Gegensatz dazu ist die einzig zulässige Operandenlänge für ein Speicherfeld die Wortlänge, wobei der maximale Verschiebungsfaktor ein Bit betragen darf.

Ferner beruht der Unterschied zwischen den logischen und der arithmetischen Verschiebung auf der Tatsache, daß das Bit V bei arithmetischen Befehlen gesetzt und bei logischen Verschiebungen gelöscht wird.

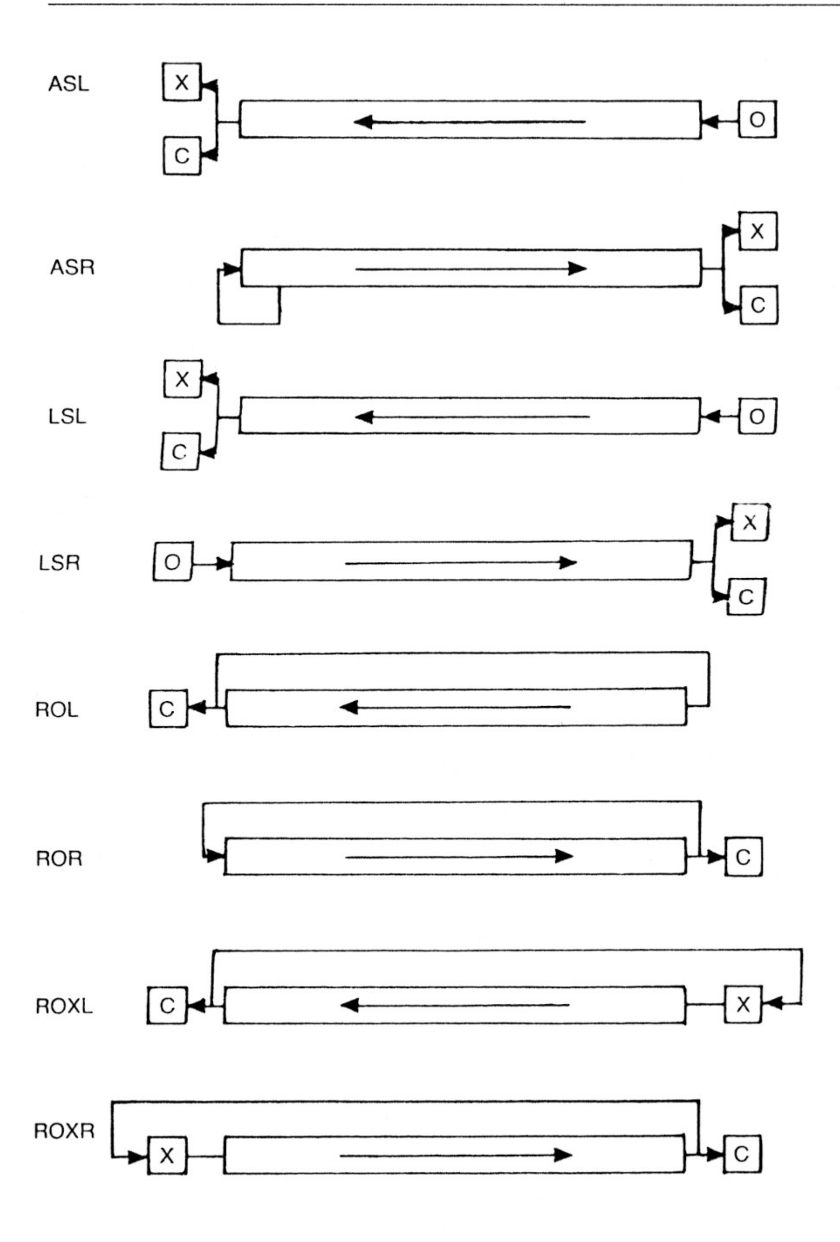


Abb. 5.12: Verschiebung und Ringverschiebung

Wenn mehrere Verschiebungen eines Speicherwortes durchgeführt werden sollen, gibt es zwei Möglichkeiten:

– Erste Möglichkeit

```
       / ROL A(0)    benötigt 8+4 Zyklen
      |  ROL A(0)
      |  ROL A(0)
      |  .
N mal <  .
      |  .
      |  .
      |  .
       \ ROL A(0)
```

Die Ausführungszeit beträgt 12*N Zyklen.

– Zweite Möglichkeit

```
MOVE (A0),D0  ; benötigt 8 Zyklen
ROL   #$N,D0  ; benötigt 6+2*N Zyklen
MOVE D0,(A0)  ; benötigt 8 Zyklen
```

Die Ausführungszeit beträgt 22+2*N Zyklen.

Setzt man

$$22+2*N<12*N$$

dann ist für N>2.2 die zweite Methode die schnellere.

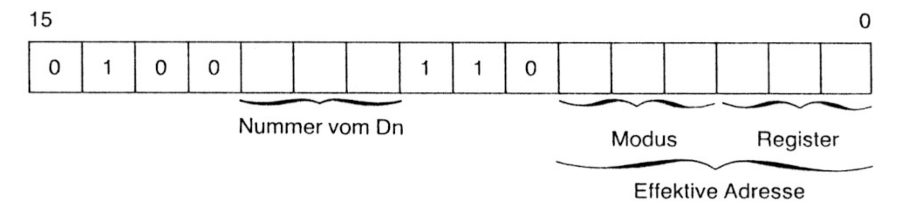
DER BEFEHL CHK

Assemblersyntax: CHK <EA>,Dn

Beschreibung:

Der Befehl CHK ermöglicht eine Überprüfung, ob der durch die effektive Adresse angegebene Wert einem Intervall #(0,(Dn)#) angehört. Ist der Wert kleiner als 0 oder größer als (Dn), so wird die Ausnahme CHK erzeugt (Vektornummer 6).

Befehlsformat:



Beispiel:

Wir wollen noch einmal das Beispielprogramm für den Befehl DBcc betrachten, das eine Fehlersuchroutine nach dem Kopieren eines Speicherbereichs in einen anderen darstellt. Nach dem Befehl DBNE wird der Wert von D0 überprüft und gemäß diesem Inhalt entschieden, ob die Kopie fehlerfrei ist oder nicht.

Das Programm sieht folgendermaßen aus:

```
GRENZE    EQU      9
          LEA      $2000, A1
          LEA      $3000, A2
          MOVEQ    # GRENZE, D0
SCHLEIFE
          CMPM     (A1) +, (A2) +
          DBNE     D0, SCHLEIFE
          CHK      # GRENZE, D0
          Folgebefehle: Routine zur Fehlerkorrektur
          .
          .
          .
          .
          END
```

Wenn alles in Ordnung ist und der Schleifenzähler D0 schließlich −1 beträgt, beginnt die Ausnahmeverarbeitung CHK, die beispielsweise eine Routine „Fehlerfreie Kopie“ einleiten kann.

Wenn D0 von −1 verschieden ist, ist also ein Fehler aufgetreten, und der nächstfolgende Befehl in einer Befehlssequenz für die Fehlerbehandlung wird ausgeführt.

DER BEFEHL MOVEM

Assemblersyntax: MOVEM <Registerliste>,<EA> oder MOVEM <EA>,<Registerliste>

Beschreibung:

Die Registerliste wird von der oder zur angegebenen effektiven Adresse transportiert.

Ein Register wird dann transferiert, wenn das entsprechende Bit im Erweiterungswort gesetzt ist: Registermaske (wird vom Assembler eingerichtet).

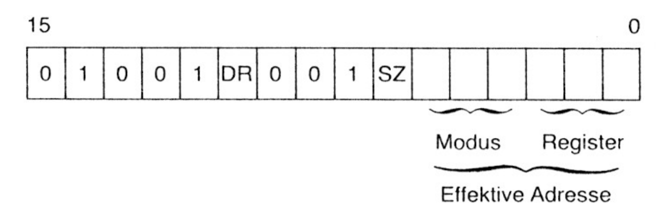
Im Fall, daß ein Wort in ein Register übertragen wird, findet eine vorzeichenbehaftete Erweiterung des Wortes statt.

Wenn Register zu einem Speicherbereich transferiert werden sollen, sind folgende Adressierungsarten zugelassen:

- Adreßregister indirekt
- Adreßregister indirekt mit Predekrementierung
- Adreßregister indirekt mit Adreßdistanzwert
- Adreßregister indirekt mit Index
- absolute Adressierung

Für den umgekehrten Transport eines Speicherbereichs in Register gelten die folgenden Adressierungsarten:

- Adreßregister indirekt
- Adreßregister indirekt mit Postinkrementierung



Wenn DR = 0 → Übertragung von Registern in Speicherbereich
Wenn DR = 1 → Übertragung von Speicherbereich in Register

Wenn SZ = 0 → Übertragung von Worten
Wenn SZ = 1 → Übertragung von Doppelworten

Abb. 5.13: Format des Befehls MOVEM

- Adreßregister indirekt mit Adreßdistanzwert
- Adreßregister indirekt mit Index
- absolute Adressierung
- relative Adressierung
- relative Adressierung mit Index

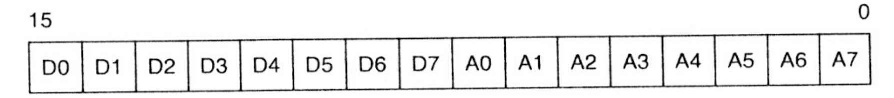
Das Erweiterungswort mit der Registermaske folgt immer auf das Befehlswort.

Adreßregister indirekt mit Predekrementierung

In diesem Modus ist nur die Registerübertragung zum Speicher gestattet. Der Inhalt der Register wird an der predekrementierten effektiven Adresse abgelegt und in Richtung der abfallenden Adressen gespeichert. Man überträgt zuerst die Adreßregister A7, A6, ..., A0, danach die Datenregister D7, D6, ..., D0.

Entsprechend dem niederwertigsten Bit des Wortes, das das Maskenregister enthält, wird das erste Register übertragen. Das höchstwertige Bit steht für das letzte zu übertragende Register.

Das Erweiterungswort hat also folgende Struktur:

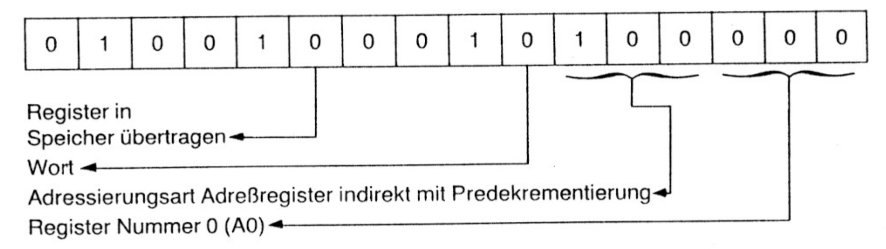


Beispiel:

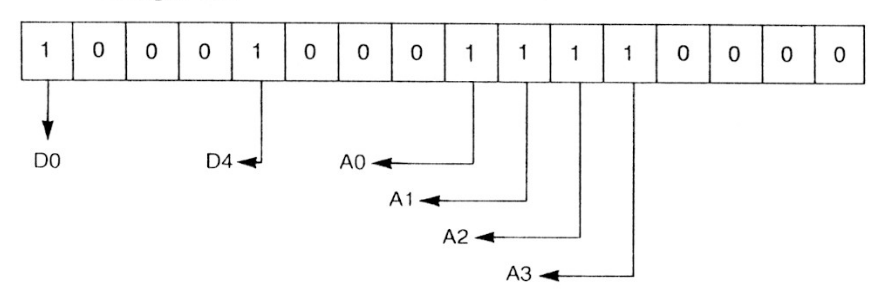
MOVEM D0/D4/A0–A3,–(A0)

Zu übertragende Register: D0, D4, A0, A1, A2, A3.

Befehlscode:



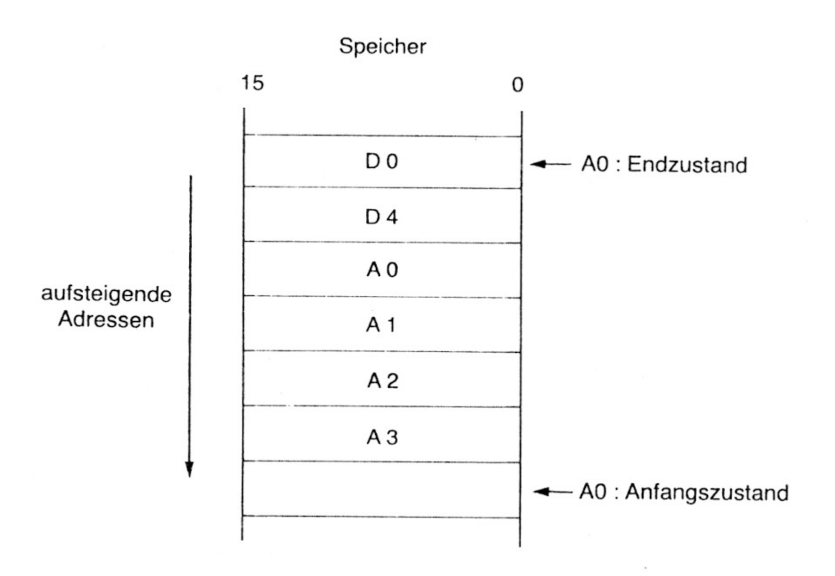
Erweiterungswort:



Hexadezimalcode:

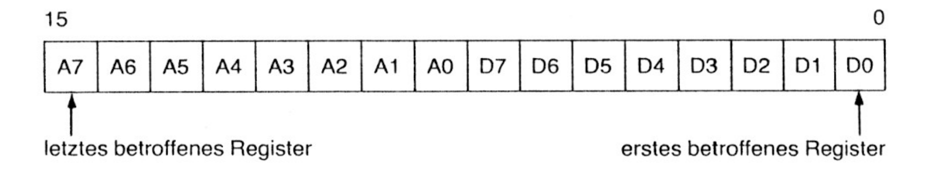
49A0
88F0

Transfer-Reihenfolge:



Adreßregister indirekt mit Postinkrementierung

In diesem Modus ist nur die Speicherfeldübertragung zu den Registern gestattet. In die Register wird der Inhalt der durch die effektive Adresse angegebenen Speicherstellen geladen. Danach wird die Adresse hochgezählt. Die Übertragungsreihenfolge in Richtung der Register ist folgende:

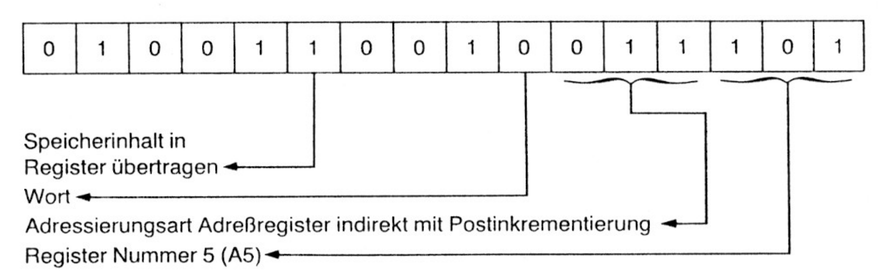


Beispiel:

MOVEM (A5)+,A0–A4/D0–D7

Zu übertragende Register: A0, A1, A3, A4, D0–D7

Befehlscode:



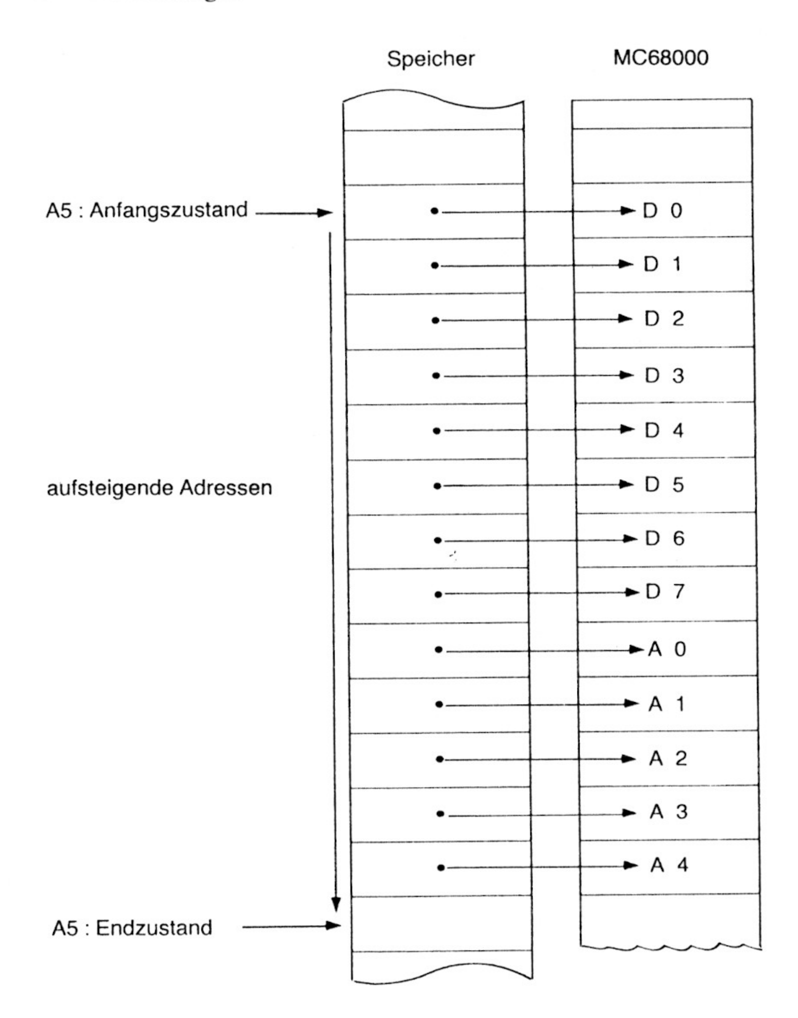
Erweiterungswort:

0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

Hexadezimalcode:

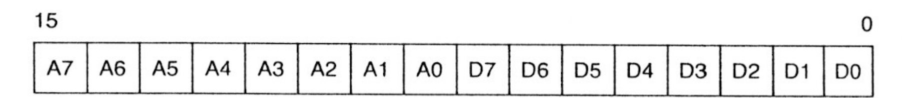
4C9D
1FFF

Transfer-Reihenfolge:



Die anderen Adressierungsarten

Die Registerinhalte werden ab der angegebenen Stelle in aufsteigender Adreßfolge im Speicher abgelegt. Die Register selbst werden dann in der gleichen Reihenfolge ausgelesen wie bei der Adressierung mit Postinkrementierung.

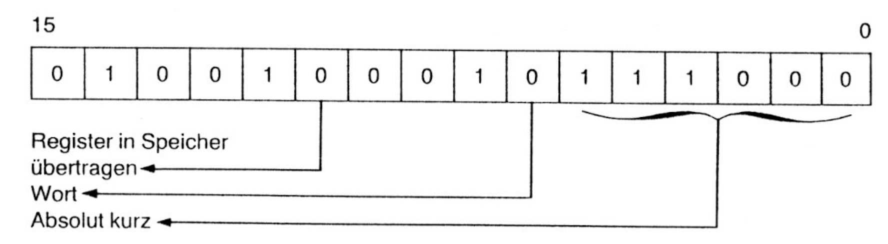


15															0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

Beispiel:

MOVEM A0/D0/D1,$3000

Befehlscode:



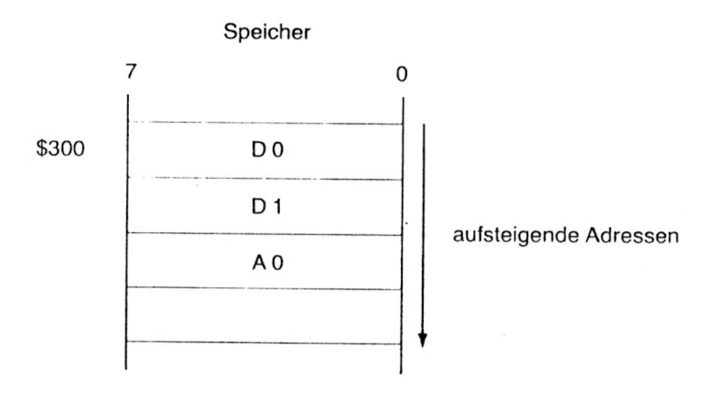
Erweiterungswort:

0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1

Hexadezimalcode:

48B8
0103
3000

Transfer-Reihenfolge:



DER BEFEHL MOVEP

Assemblersyntax: MOVEP Dx,D(Ay) oder MOVEP D(Ay),Dx

Beschreibung:

Die Übertragung findet zwischen dem Datenregister und den Bytes des Speichers mit abwechselnd geraden und ungeraden Adressen statt, und zwar beginnend mit der im Befehl angegebenen Adresse (Adreßregister indirekt mit Adreßdistanzwert).

Das obere Byte des Datenregisters wird zuerst übertragen, das untere zuletzt.

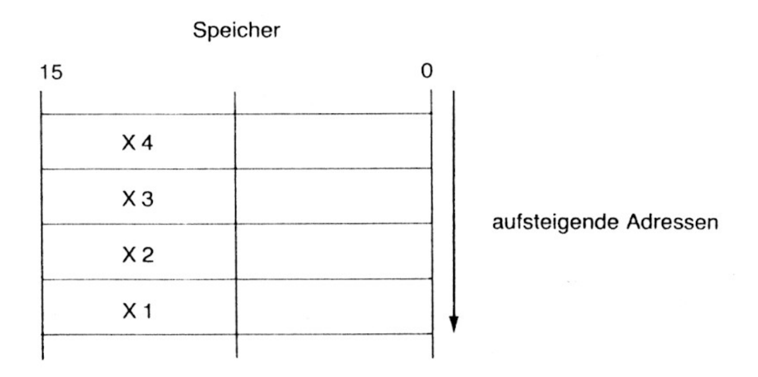
Beispiel:

Transfer eines Doppelwortes zum oder vom Speicher an einer geraden Adresse.

Einteilung des Registers:

31 24	23 16	15 8	7 0
X4	X3	X2	X1

Einteilung des Speichers:

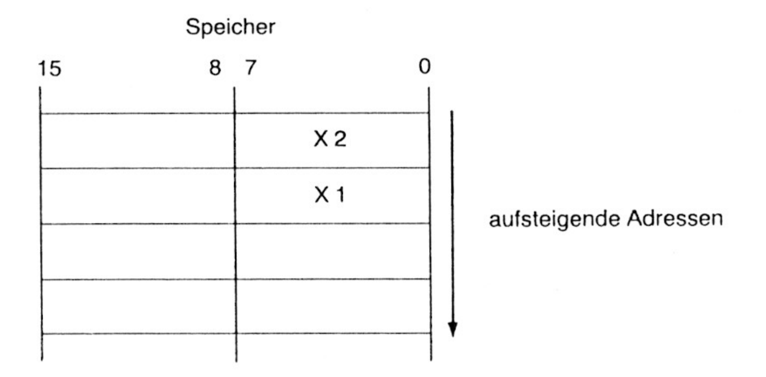


Transfer eines Wortes zum oder vom Speicher an einer ungeraden Adresse.

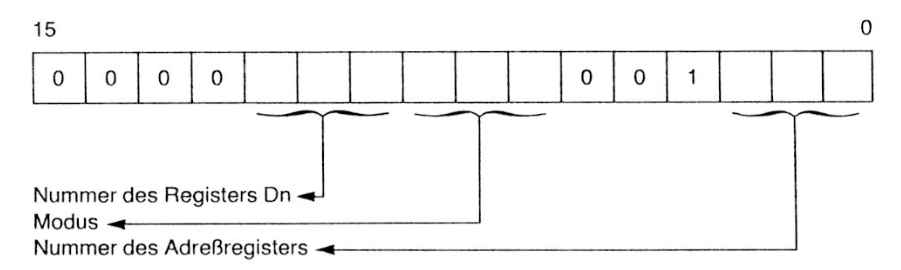
Einteilung des Registers:

31 … 24	23 … 16	15 … 8	7 … 0
		X 2	X 1

Einteilung des Speichers:



Befehlsformat:



Der Modus gibt die Operationsrichtung und die -länge an:

Modus=100; Transfer eines Wortes vom Speicher zu einem Register.
Modus=101; Transfer eines Doppelwortes vom Speicher zu einem Register.
Modus=110; Transfer eines Registerwortes zum Speicher.
Modus=111; Transfer eines Registerwortes doppelter Länge zum Speicher.

Dem Befehlswort folgt unmittelbar das Wort, das den Verschiebungsfaktor enthält.

Dieser Befehl erlaubt auf einfache Weise, die Peripheriegeräte zu programmieren, deren Registernummern aufeinanderfolgenden geraden oder ungeraden Adressen entsprechen.

Wir nehmen uns jetzt als einfaches Programmierbeispiel den peripheren Baustein PIA 6821 (Parallel Interface Adapter) vor.

Das Funktionsschema des 6821 ist in Abb. 5.14 dargestellt.

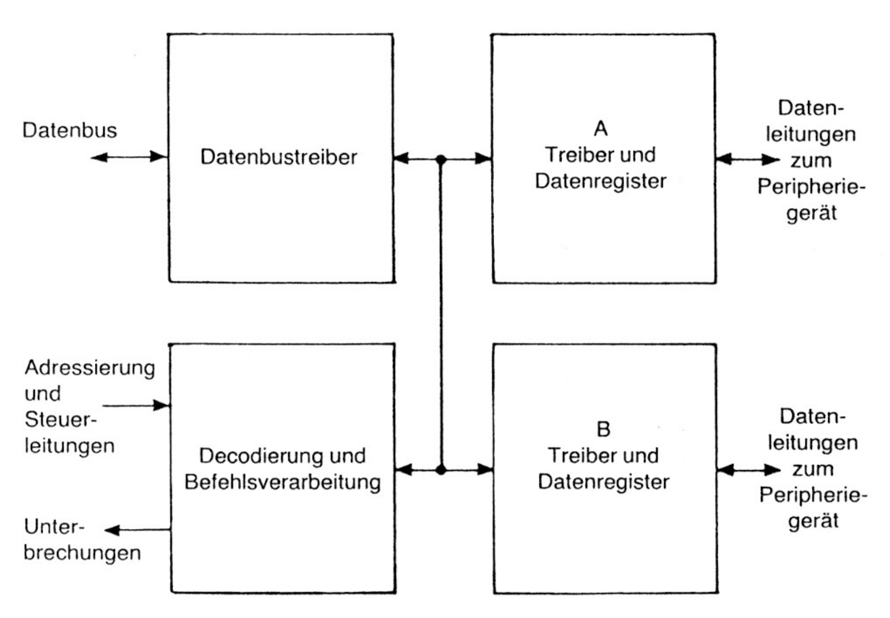


Abb. 5.14: Schematischer Aufbau des 6821

Die Abb. 5.15 verdeutlicht die interne Organisation des Bausteins.

Sechs interne Register des Bausteins 6821 sind von außen zugänglich:

- zwei Peripheriedatenregister PRA und PRB
- zwei Datenrichtungsregister DDRA und DDRB
- zwei Steuerregister CRA und CRB.

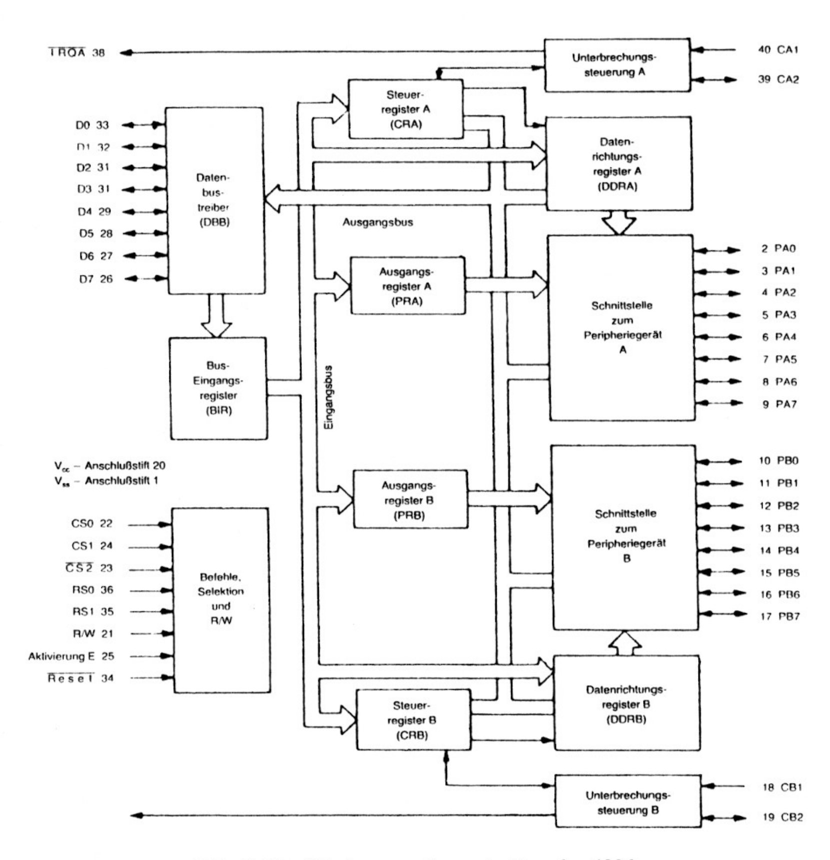


Abb. 5.15: Die interne Organisation des 6821

Die Leitungen der beiden Ports A und B können unabhängig als Ein- oder Ausgänge programmiert werden, in Abhängigkeit von den Datenrichtungsregister betrachtet werden. Eine 1 (bzw. eine 0) in einem der Bits des Registers programmiert die entsprechende Portleitung als Ausgang (bzw. als Eingang).

Die beiden Steuerregister CRA und CRB steuern die vier Leitungen CA1, CA2, CB1, CB2. Sie legen auch den Zugriff auf die Peripheriedatenregister und die Datenrichtungsregister fest.

Die interne Adreßstruktur und das Format der Befehlsworte sind aus den Tabellen der Abb. 5.16 und 5.17 ersichtlich.

RS1	RS0	Bit in Befehls-register CRA2	CRB2	ausgewähltes Register
0	0	1	x	Peripheres Datenregister A
0	0	0	x	Datenrichtungsregister A
0	1	x	x	Steuerregister A
1	0	x	1	Peripheres Datenregister B
1	0	x	0	Datenrichtungsregister B
1	1	x	x	Steuerregister B

x = ohne Bedeutung

Abb. 5.16: Interne Adressierung

	7	6	5	4	3	2	1	0
CRA	IRQA1	IRQA2	Befehl von CA2			Zugriff auf DDRA	Befehl von CA1	

	7	6	5	4	3	2	1	0
CRB	IRQB1	IRQB2	Befehl von CB2			Zugriff auf DDRB	Befehl von CB1	

Abb. 5.17: Format der Befehlsworte

Wir wollen die Programmierung dieses Bausteins vereinfachen, indem wir folgendermaßen initialisieren:

- Port A
  - die 4 unteren Bits als Ausgang
  - die 4 oberen Bits als Eingang
- Port B
  - alle 8 Bits als Ausgang

Unterbrechungssteuerung nicht aktiv.
Zugriff zu Datenregistern PRA, PRB.

Die Adressen der Register sind:

ADA: DDRA/PRA: $1001
ADB: DDRB/PRB: $1003
ADRA: CRA: $1005
ADRB: CRB: $1007

Eine Möglichkeit, den Baustein zu programmieren, ist nun die folgende:

```
ADA     EQU     $1001
ADB     EQU     ADA + 2
```

```
ADRA    EQU       ADA + 4
ADRB    EQU       ADA + 6
        MOVE.B    # $0F, ADA
        MOVE.B    # $FF, ADB
        MOVE.B    # $04, ADRA
        MOVE.B    # $04, ADRB
```

Die zweite Möglichkeit ergibt sich aus der Verwendung des MOVEP-Befehls:

```
ADA     EQU       $1001
        LEA       ADA, A0
        MOVE.L    # $0FFF04404, D0
        MOVEP.L   D0, 0(A0)
```

Mit dieser Programmierungsmethode kann man gleichzeitig N Peripheriebausteine initialisieren, wenn die Initialisierungswerte so im Speicher abgelegt sind, wie es die Abb. 5.18 zeigt.

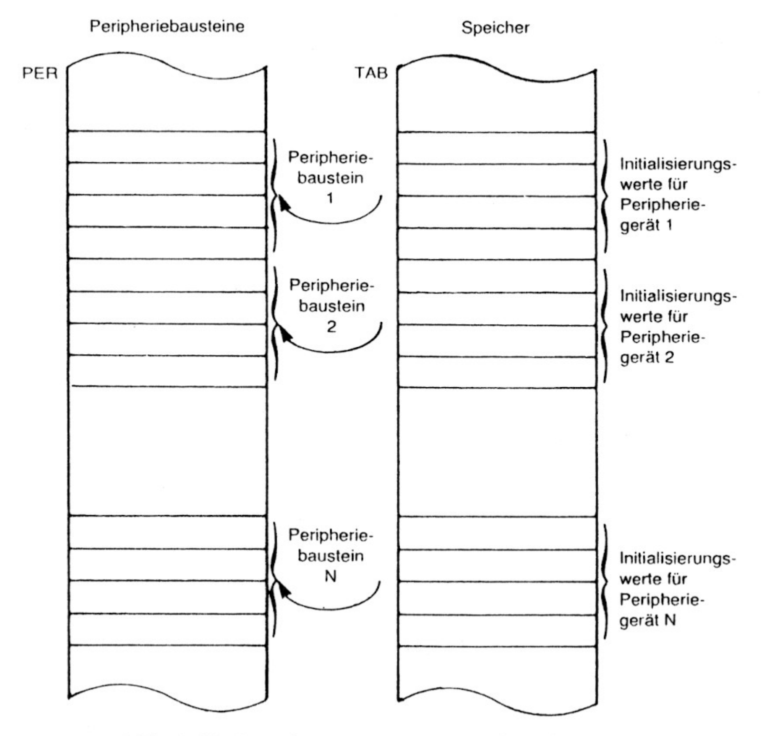


Abb. 5.18: Initialisierung von N Peripheriebausteinen

Das Programm sieht so aus:

```
         MOVEQ    #N,D1          ;D1 enthält die Anzahl des
                                 ;zu initialisierenden
                                 ;Peripheriebausteins
         LEA      ADPER1, A0     ;A0 enthält die Adresse
                                 ;des ersten Peripherie-
                                 ;bausteins
         LEA      TABLE, A1      ;A1 enthält die Startadresse
                                 ;der Tabelle
         BRA      INIT           ;D0 wird von N auf −1 her-
                                 ;untergezählt. Dies ermög-
                                 ;licht die Initialisierung von
                                 ;N Peripheriebausteinen
SCHLEIFE MOVE.L   (A1)+, D0      ;Initialisierungswert in (D0)
                                 ;schreiben und A1 erhöhen,
                                 ;so daß es auf das folgende
                                 ;Wort zeigt
         MOVEP.L  D0, 0(A0)      ;Übertragung zum Periphe-
                                 ;riebaustein
         ADD.L    #8, A0         ;Setzen von A0 auf den
                                 ;nächsten zu initialisieren-
                                 ;den Peripheriebaustein
INIT     DBRA     D0, SCHLEIFE   ;nächste Initialisierung
         END
```

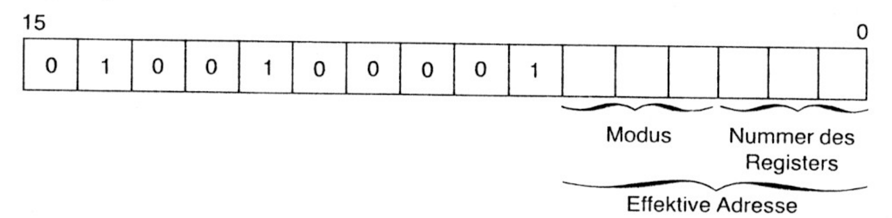
DER BEFEHL PEA

Assemblersyntax: PEA <EA>

Beschreibung:

Die effektive Adresse wird berechnet und auf den Stapel gelegt. Der Operand ist 32 Bits lang.

Befehlsformat:

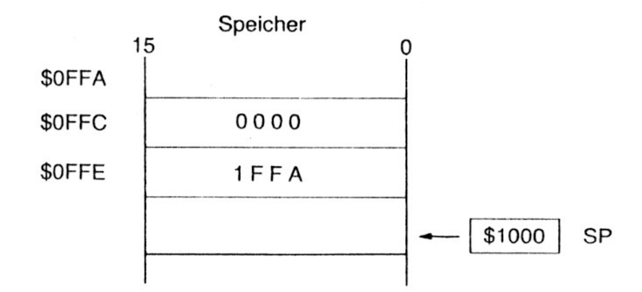


Beispiel:

```
LEA  $2000,A0
LEA  $1000,SP
PEA  -6(A0)
END
```

Ausführung:

Die Adresse $1FFA (also $2000−6) ist die effektive Adresse und wird ab der Adresse $0FFC auf den Stapel gelegt.



DER BEFEHL TAS

Assemblersyntax: TAS <EA>

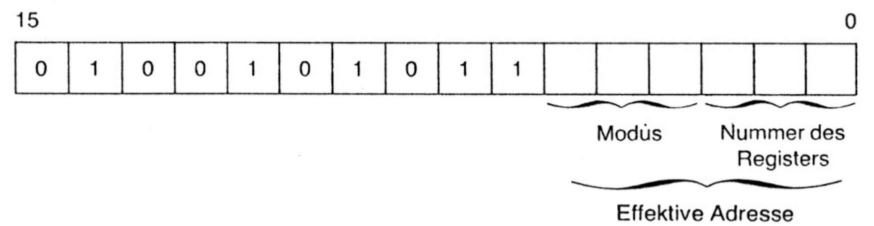
Beschreibung:

Der Operand, der Bytelänge hat, wird getestet, bevor N und Z entsprechend gesetzt werden.

Das höchstwertige Bit des Bytes wird immer auf 1 gesetzt.

Dieser Befehl löst einen Lese-/Änderungs-/Schreibvorgang aus, der nicht unterbrechbar ist.

Befehlsformat:



Dieser Befehl gewinnt seine Bedeutung in Multiprozessor-Systemen.

Bei diesen Systemen haben stets verschiedene Prozessoren Zugriff zu gemeinsamen Ressourcen. Diese Ressourcen können Arbeitsspeicher oder Peripheriebausteine sein.

Zwei Beispiele für Multiprozessor-Systeme sind in Abb. 5.19 und 5.20 gezeigt.

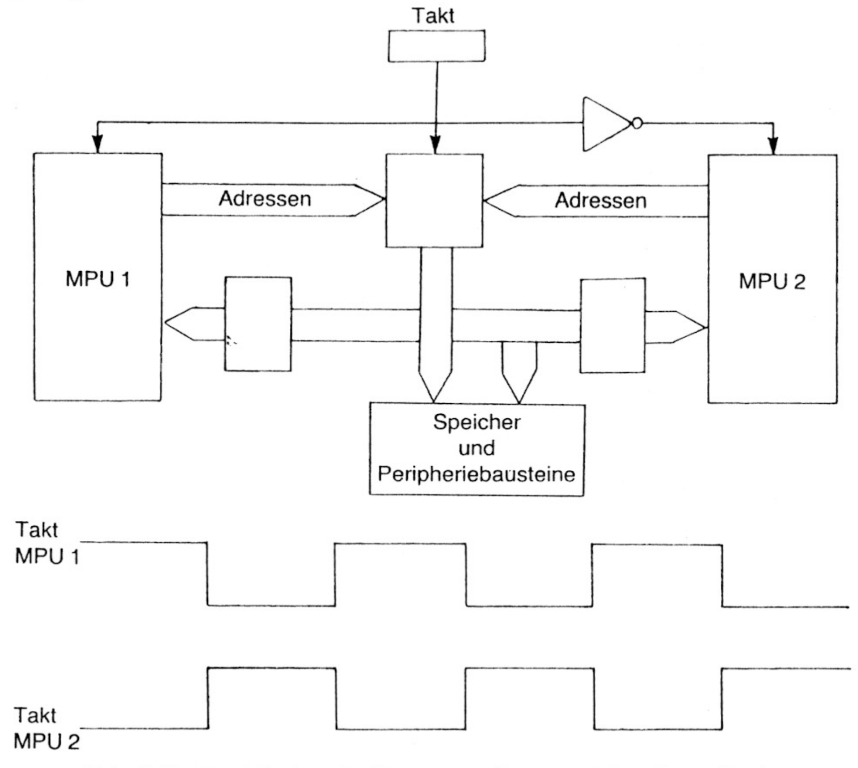


Abb. 5.19: Zweiphasiges Multiprozessor-System mit Speicherverbindung

In der ersten Konfiguration arbeiten die Prozessoren taktverschieden, und es können auch nicht mehr als zwei sein. Das Multi-Bus-System eröffnet die Möglichkeit, zwei oder auch mehr Prozessoren miteinander zu verbinden.

Bei der zweiten Konfiguration könnte man sich beispielsweise ein Verbindungsschema zwischen dem Mikroprozessor 68000 und dem IPC 68120/21

(Intelligent Peripheral Controller), einem universellen, intelligenten Peripherie-Steuerbaustein, für die Aufnahme und Verarbeitung analoger Daten vorstellen (Abb. 5.21).

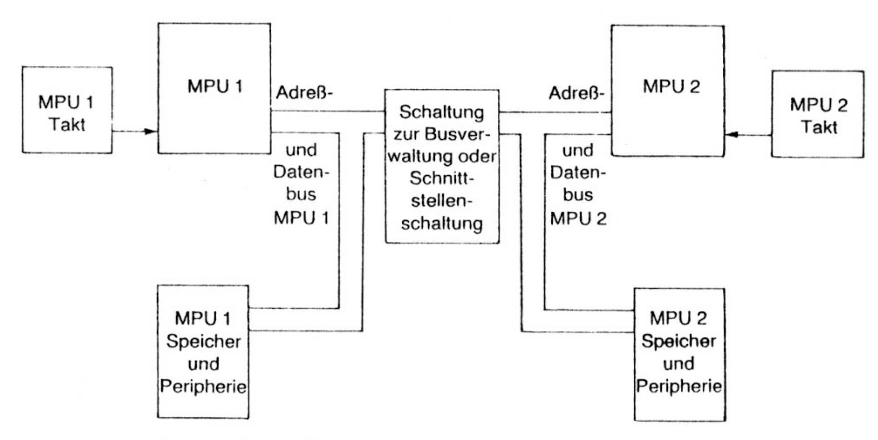


Abb. 5.20: Multiprozessor-System mit Multi-Bus-Technik

Der MC68120 ist ein Mikroprozessor, der die nötige Hardware mitbringt, um mehrere Prozessoren in einem System zu verkoppeln.

Dazu gehören vor allen Dingen ein RAM mit Doppelzugriff und Semaphore-Register. (Eine Semaphore ist ein Kennzeichen, das anzeigt, ob ein bestimmtes gemeinsam benutztes Gerät frei ist oder nicht.) Das RAM mit Doppelzugriff gewinnt seine Bedeutung hauptsächlich durch Übertragung zwischen den Systembausteinen und dem Peripherie-Steuerbaustein, damit die lokalen Schaltungen nicht gestört werden.

Die sechs Semaphore-Register werden benutzt, um die Datenzugriffskontrolle zwischen dem Systembus und den lokalen Bussen zu regeln. Es wird auf diese Weise eine Zusammenarbeit der verschiedenen Prozessoren geregelt, bei der keine Synchronisierungsprobleme auftreten.

Das Semaphore-Register ist ein 8-Bit-Register, dessen höchstwertiges Bit man „Semaphore-Bit" nennt. Es ist dasjenige, das die Aktivität der jeweiligen Einheit meldet.

- Ist das Bit 0, so steht die Einheit zur Verfügung.
- Ist das Bit 1, so wird die Einheit bereits anderweitig benutzt.

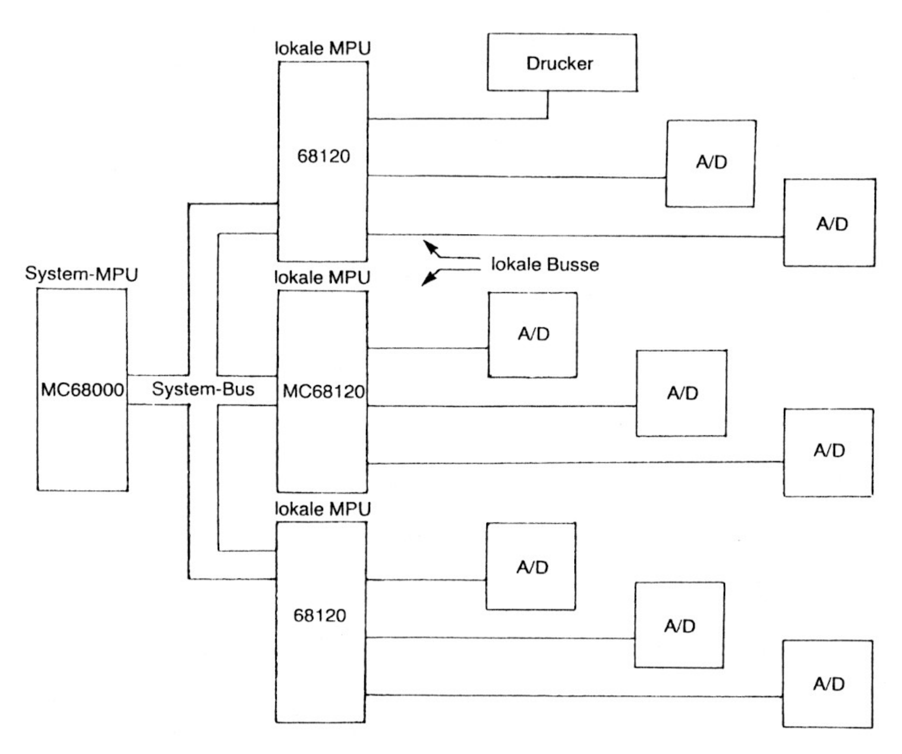


Abb. 5.21: Verbindung zwischen einem MC68000- und drei MC68120-Bausteinen (schematisch)

In einigen Semaphore-Registern wird das Bit 6 (ownership) nur beim Lesen verwendet, und es gibt an, welcher Prozessor das Semaphore-Bit gesetzt hat.

Wenn das Semaphore-Bit aktiv ist, gibt das Bit 6 an, welcher Prozessor es gesetzt hat.

Wenn das Semaphore-Bit 0 ist, gibt Bit 6 den letzten Prozessor an, der es gesetzt hatte.

Der TAS-Befehl gestattet das Abfragen und Setzen des Semaphore-Bits.

Das Ablaufdiagramm in Abb. 5.22 zeigt den Fall, daß verschiedene Prozessoren den Zugriff zu einem gemeinsamen Speicherbereich, der aus mehreren Blöcken besteht, wünschen.

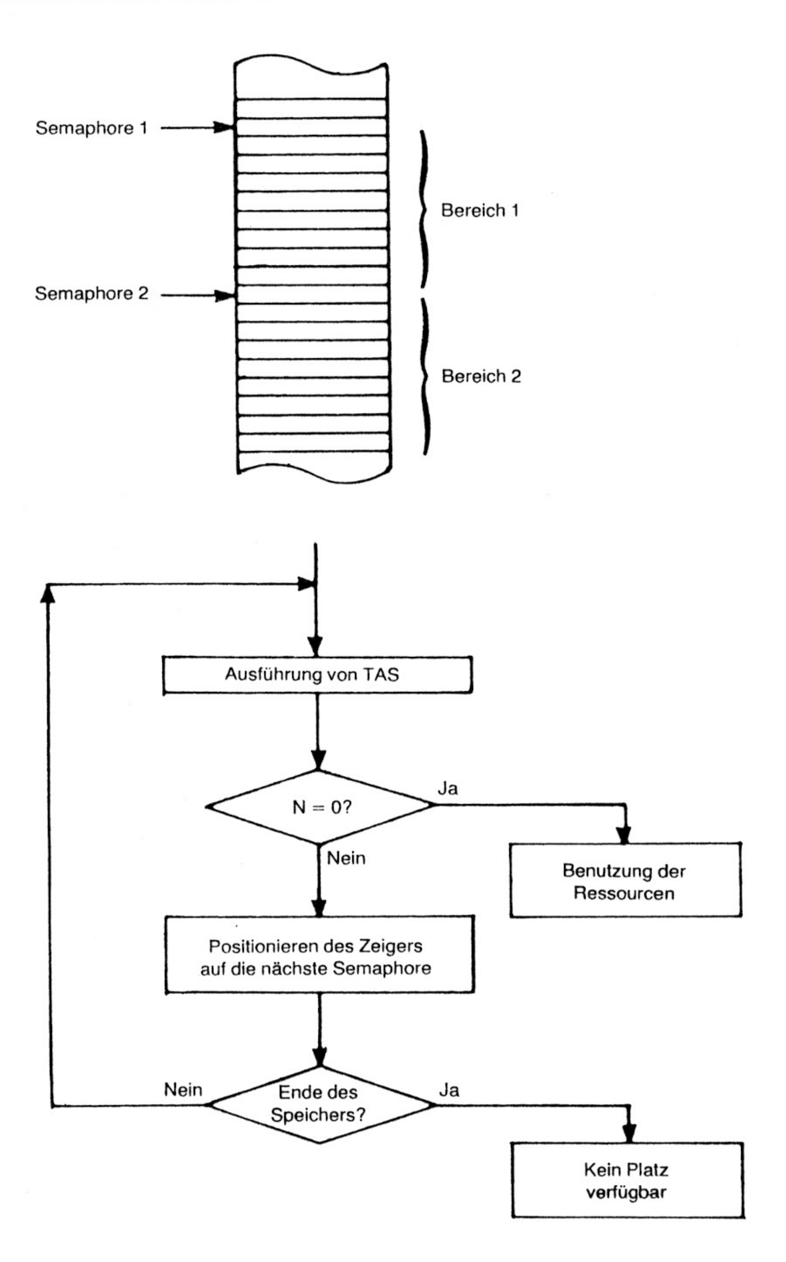


Abb. 5.22: Beispiel für die Verwendung des TAS-Befehls

Es ist notwendig, daß der TAS-Befehl in einem nicht unterbrechbaren Zyklus abgewickelt wird, damit ein Fall, wie er in Abb. 5.23 aufgezeigt wird, nicht eintreten kann.

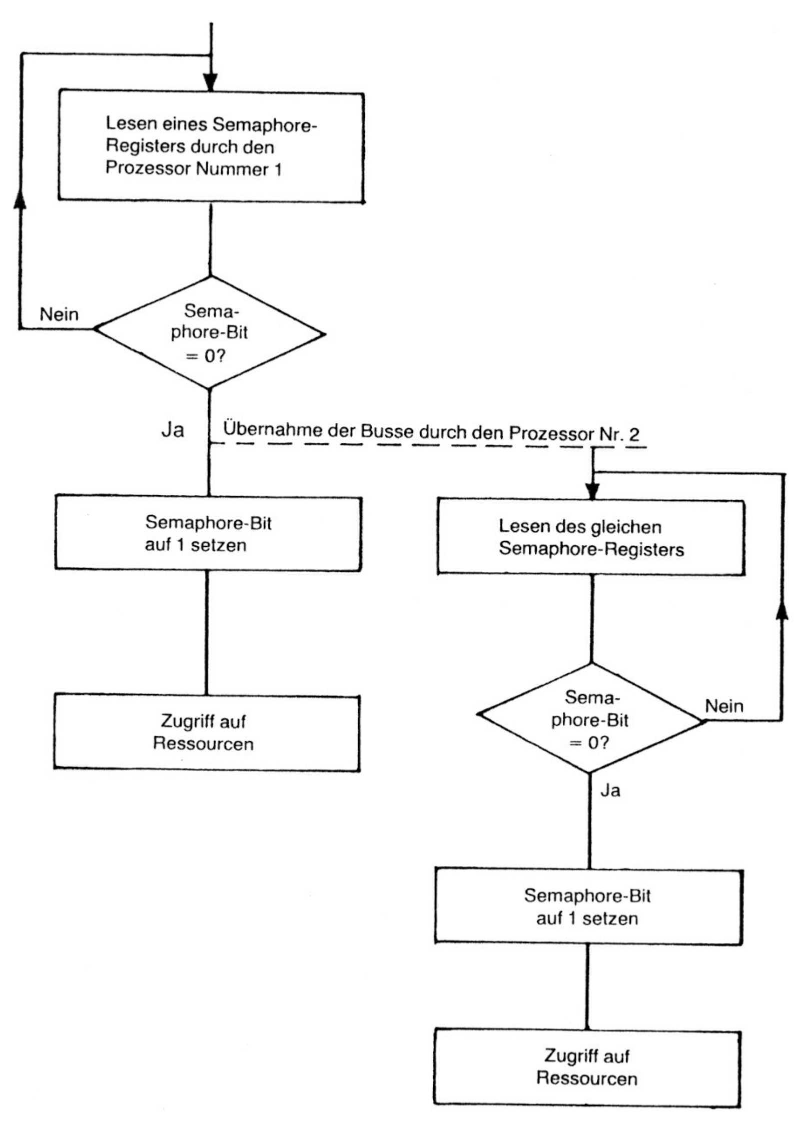


Abb. 5.23: Zwei Prozessoren arbeiten mit demselben Betriebsmittel

DIE BEFEHLE FÜR BITMANIPULATIONEN: BTST, BSET, BCLR, BCHG

BTST

Assemblersyntax: BTST Dn,<EA> oder BTST #xxx,<EA>

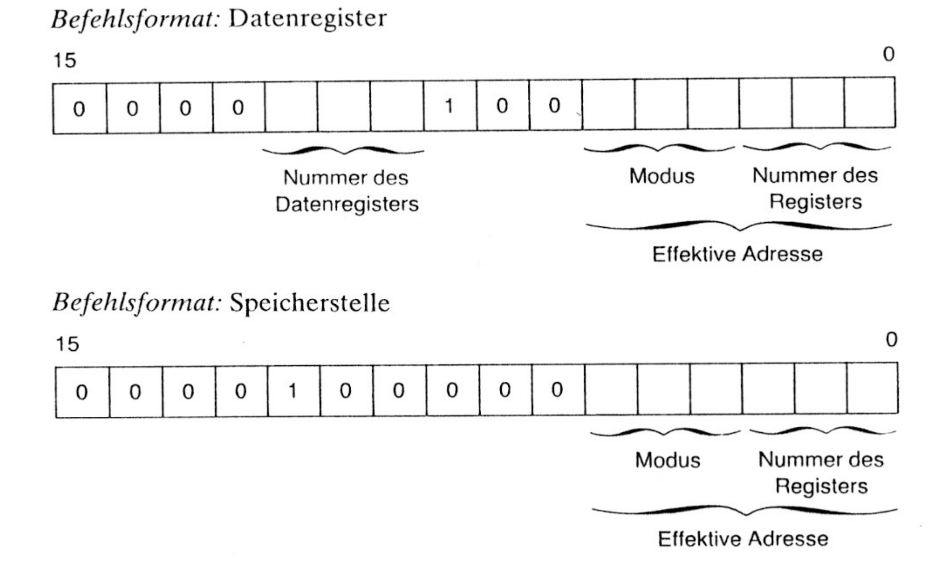
Beschreibung:

Ein Bit des Zieloperanden wird getestet. Das Ergebnis dieses Tests wird im Bit Z sichtbar gemacht.

Die Nummer des zu testenden Bits wird entweder unmittelbar oder in einem Datenregister angegeben.

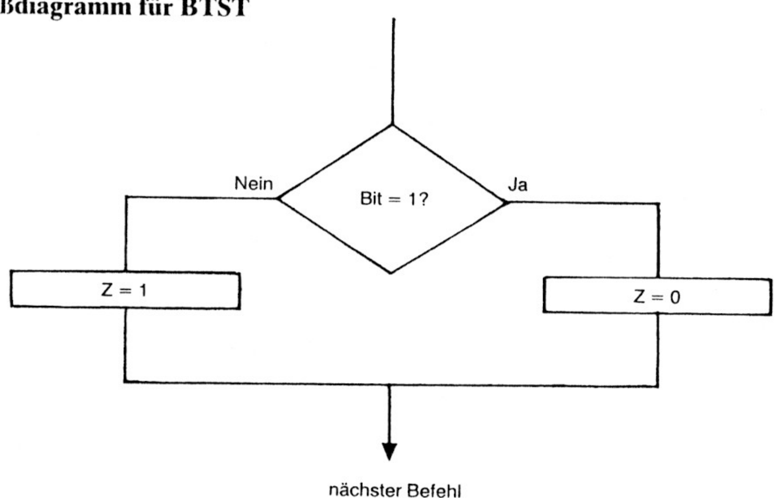
Wenn die effektive Adresse ein Datenregister ist, kann man die Bits 0, 1, ..., 31 testen.

Im Gegensatz dazu können bei einem Speicherbereich nur jeweils die Bits 0 bis 7 getestet werden.



Die vier Befehle können, wie in Abb. 5.24 dargestellt, schematisiert werden.

Flußdiagramm für BTST



Flußdiagramm für BSET

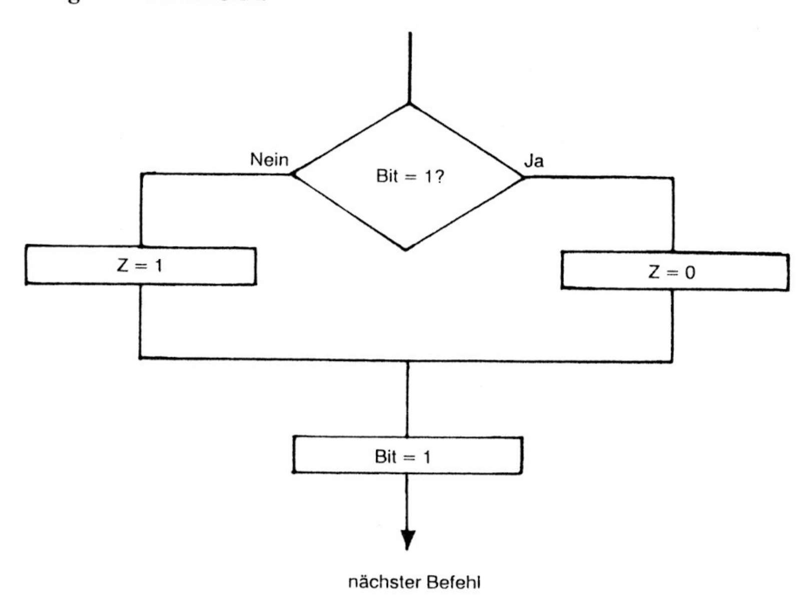


Abb. 5.24: Befehle zum Testen eines Bits

Flußdiagramm für BCLR

Flußdiagramm für BCHG

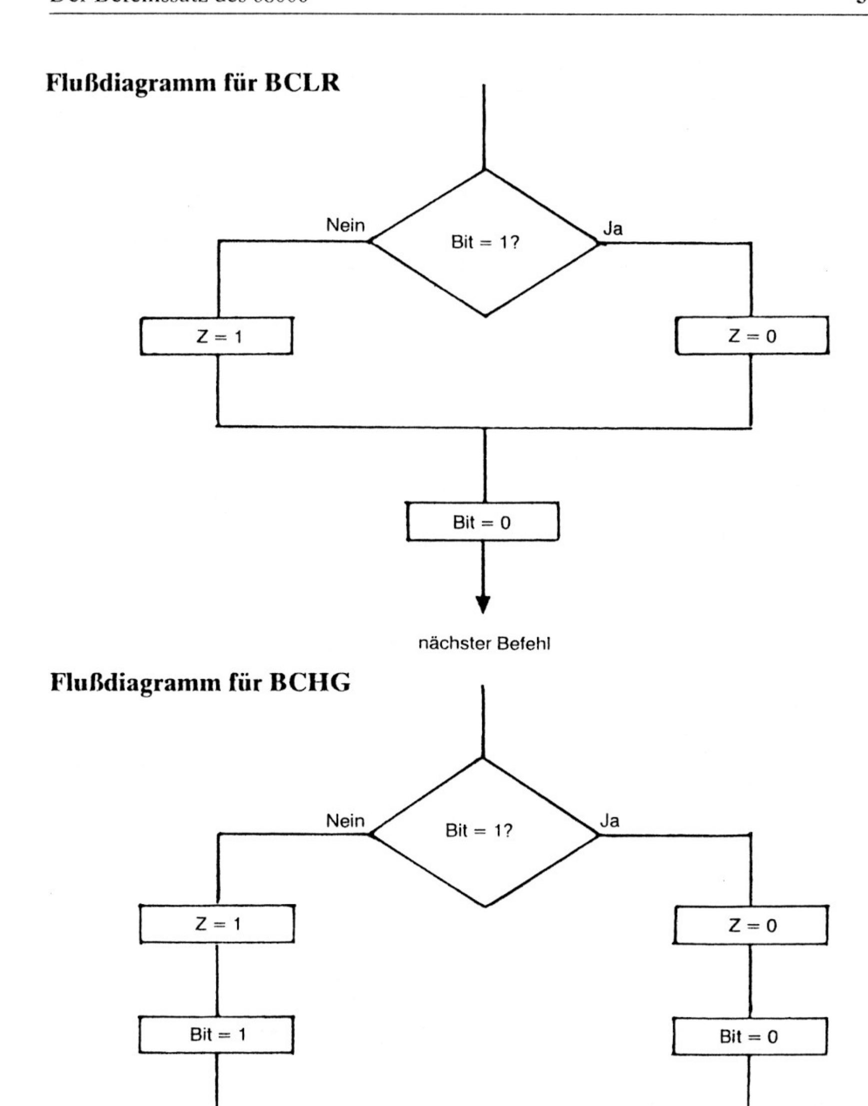


Abb. 5.24: Befehle zum Testen eines Bits (Forts.)

Beispiel:

Wir haben die folgenden Befehle:

```
                      00001000      ORG        $1000
          00001000    303C00FE      MOVE.W     # $00FE, D0
          00001004    08000000      BTST       # 0, D0
                                    END
TOTAL ERRORS 0
TOTAL WARNINGS 0
```

Der Programmlauf ergibt:

```
D0-D7            265300FE      30303033      46495820      00000000
                 20202020      20202020      00000000      00000061
A0-A7            45322020      434E3130      00000000      30303033
                 000012BD      01A0D104      00000000      00000000
PC = 00001004       SR = 8000      BTST        # 0, D0
D0-D7            265300FE      30303033      46495820      00000000
                 20202020      20202020      00000000      00000061
A0-A7            45322020      434E3130      00000000      30303033
                 000012BD      01A0D104      00000000      00000000
PC = 00001008       SR = 8004
```

DER BEFEHL LINK

Assemblersyntax: LINK An,#<Adreßdistanzwert D>

Beschreibung:

Der Befehl veranlaßt drei unterschiedliche Aktionen:

1. Der Wert des Adreßregisters wird auf den Stapel an der Adresse abgelegt, die durch den Stapelzeiger SP angegeben wird.
2. Das Adreßregister wird mit dem Inhalt des Stapelzeigers gefüllt.
3. Der Adreßdistanzwert wird auf 32 Bits erweitert und zum Inhalt des Stapelzeigers addiert.

An→–(SP)
SP→An
SP+D→SP

Das Adreßregister An wird auch Rahmenzeiger (RZ) oder Frame Pointer (FP) (auch Speicherbereichszeiger) genannt. Register, die diese Funktion wahrnehmen können, sind die Register A0, A1, ..., A6.

Der Adreßdistanzwert oder Verschiebungsfaktor ist ein 16-Bit-Datenwert:

$$-32768 \leq D \leq +32767$$

Befehlsformat:

15															0
													Nummer des Registers		
0	1	0	0	1	1	1	0	0	1	0	1	0			
Adreßdistanzwert															

Die drei Übertragungen, die durch den LINK-Befehl ausgelöst werden, sind die folgenden:

An→–(SP) : Daten oder Variablen des FP werden auf den Stapel gebracht.
SP→An : Stapelzeiger wird in den FP gelden.
SP+D→SP : Verschiebungswert wird zum SP addiert.

Wir machen uns diesen Vorgang noch einmal anhand der Abb. 5.25 klar.

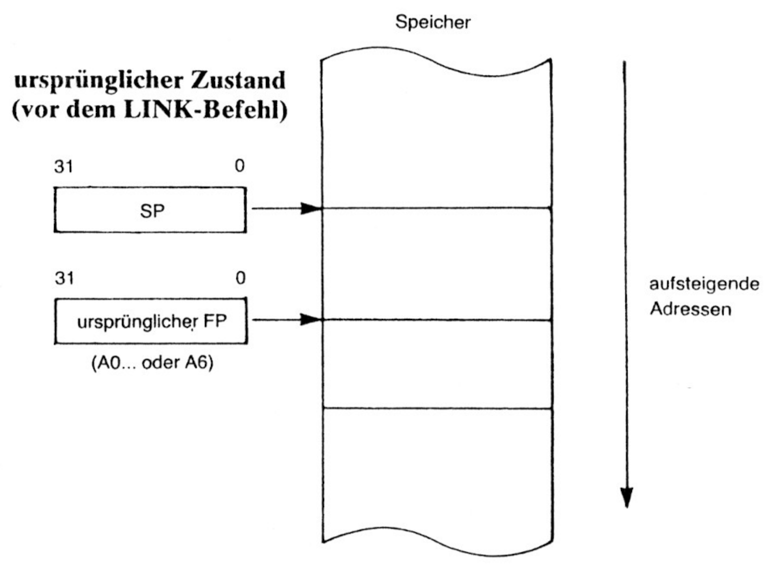
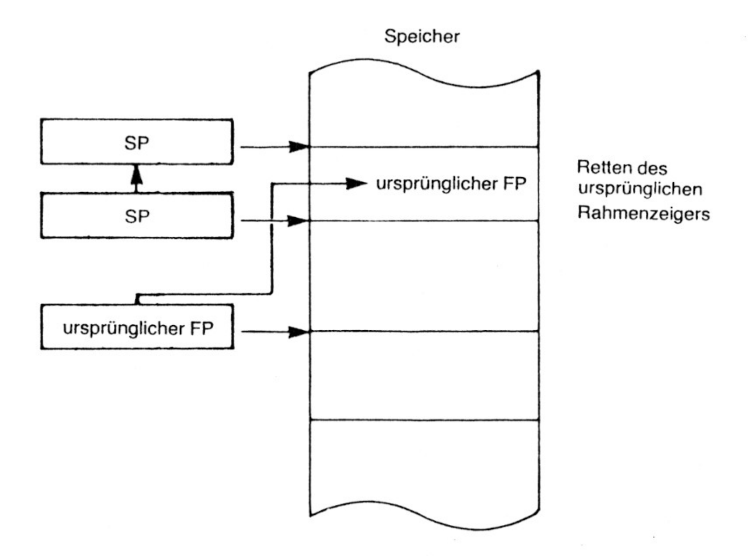


Abb. 5.25: Die Arbeitsweise des LINK-Befehls

1. Aktion



2. Aktion

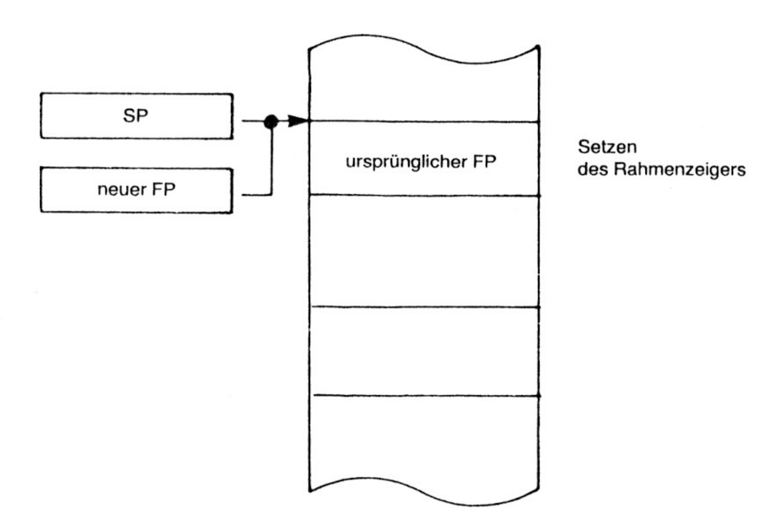


Abb. 5.25: Die Arbeitsweise des LINK-Befehls (Forts.)

3. Aktion

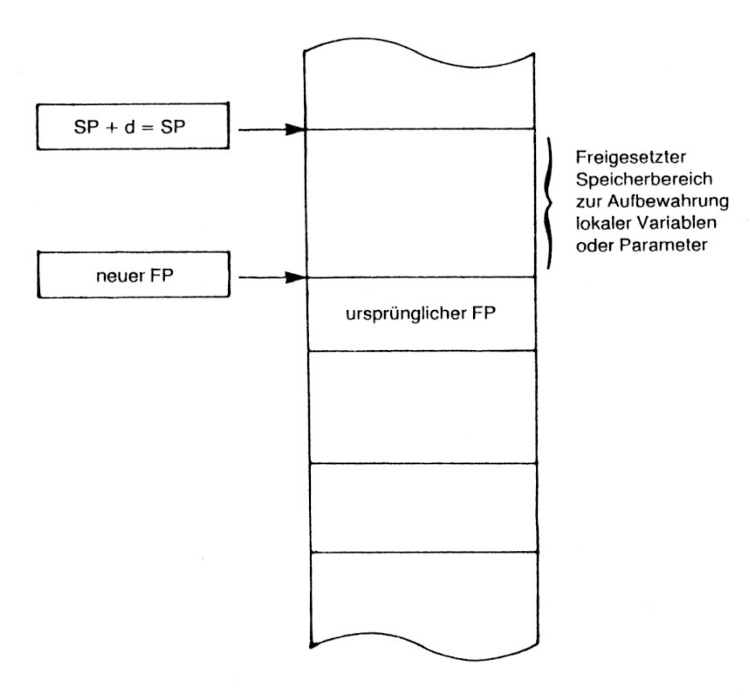


Abb. 5.26: Die Arbeitsweise des LINK-Befehls (Forts.)

DER BEFEHL UNLK

Assemblersyntax: UNLK An

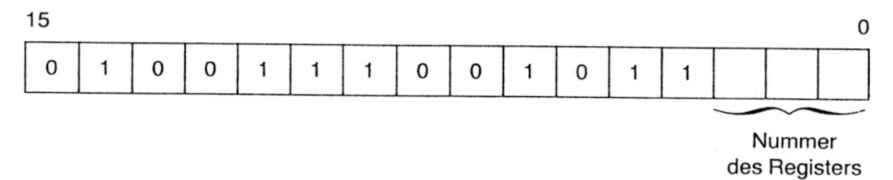
Beschreibung:

Der Inhalt des angegebenen Adreßregisters wird zum Stapelzeiger transferiert. Daraufhin wird der Inhalt des Speicherbereichs, der durch den Stapelzeiger adressiert ist, in das Adreßregister übertragen.

An→SP
(SP)→An

Das Register An fungiert hier wieder als Rahmenzeiger (Frame Pointer).

Befehlsformat:



UNLK bewirkt genau das Gegenteil von LINK. Der Speicherbereich wird wieder freigegeben und der Rahmenzeiger wiederhergestellt. Wir wollen uns noch einmal den Zustand des Stapels nach Beendigung des LINK-Befehls vor Augen führen, um den Vorgang beim Befehl UNLK näher zu untersuchen (Abb. 5.26).

ursprünglicher Zustand
(vor UNLK, nach LINK)

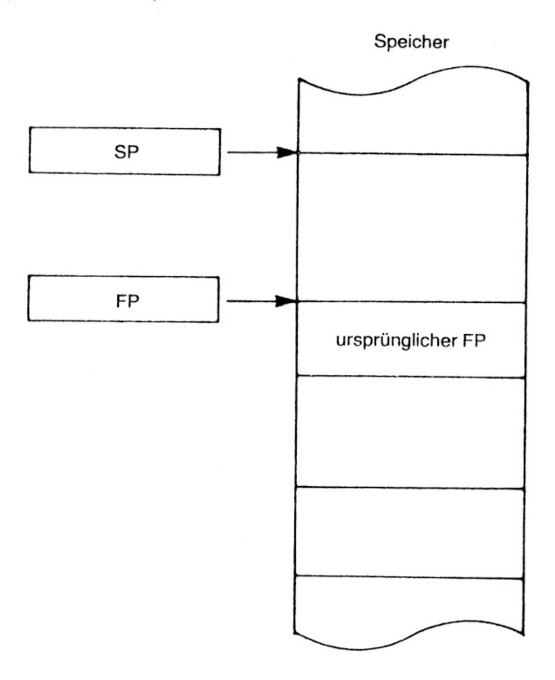


Abb. 5.26: Die Arbeitsweise des UNLK-Befehls

1. Aktion

SP

FP

ursprünglicher FP

Der Stapelzeiger adressiert den ursprünglichen Rahmenzeiger

2. Aktion

SP

ursprünglicher FP

Der Rahmenzeiger nimmt wieder seinen ursprünglichen Wert an

Abb. 5.26: Die Arbeitsweise des UNLK-Befehls (Forts.)

LINK UND UNLK

LINK und UNLK sind zwei sehr anspruchsvolle Befehle, die es ermöglichen, auf hohem Sprachniveau zu arbeiten, und die auch sehr gut Sprachen wie zum Beispiel Pascal unterstützen. Dieser Sprachtyp kennt den Begriff der Prozeduren (Unterprogramme), die dem Blockkonzept moderner Sprachen entsprechen und mit lokalen Daten arbeiten. Prozeduren verarbeiten prinzipiell zwei Sorten von Variablen:

- lokale Variablen, die nur innerhalb der Prozedur deklariert und bekannt sind,
- globale Variablen, die außerhalb des Prozedurblocks deklariert sind und in ihm und im übergeordneten Programm bekannt sind (Abb. 5.27).

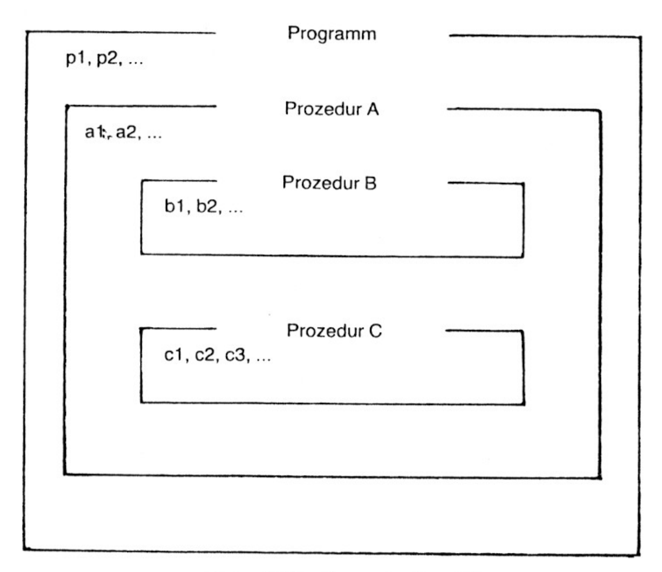


Abb. 5.27: Prozedurbegriff

Zum Beispiel sind die Variablen a1, a2, ... als lokale Variablen für die Prozedur A und global für die Prozeduren B und C anzusehen.

Wenn das Hauptprogramm die Prozedur A aufruft, die ihrerseits B und dann C aufruft, ergibt sich, daß die lokalen Variablen zum Zeitpunkt der Prozeduröffnung eingerichtet werden müssen und daß sie sofort nach dem Verlassen der Prozedur wieder freigegeben werden.

Durch Nutzung eines Stapelspeichers für das Ablegen der lokalen Daten der momentan bearbeiteten Prozedur kann auf einfache Weise ein Übergang von einer Variablenumgebung zu einer anderen verwirklicht werden (Abb. 5.28).

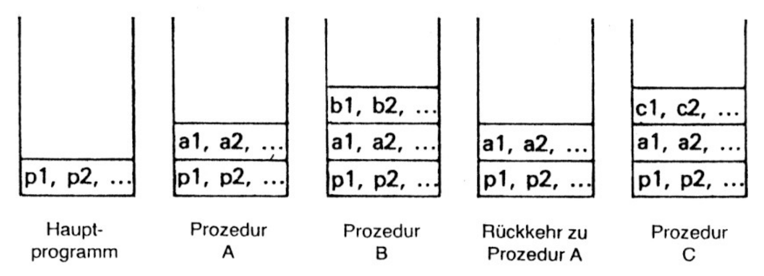


Abb. 5.28: Zustandsfolge des Stapelspeichers für sukzessive Unterprogrammaufrufe

Diese Prozedur-orientierte Programmabwicklung wird durch einen Speicherbereichszeiger unterstützt, der jeweils den Variablen einer Prozedur zugeordnet wird und der bei jedem Aufruf einer weiteren Prozedur gerettet werden muß (Abb. 5.29).

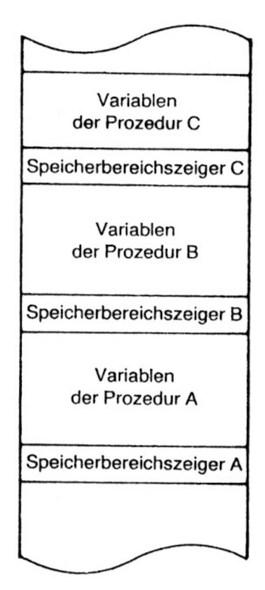


Abb. 5.29: Bedeutung des Speicherbereichszeigers

Diese Verkettung der Speicherbereichsadressen gestattet bei der Rückkehr aus untergeordneten Routinen zu aufrufenden Prozeduren eine entsprechende Wiederentnahme der zugehörigen Felder vom Stapel. Es ist ferner ersichtlich, daß die Variablen der zuletzt aufgerufenen Prozedur nicht zugleich mit deren Wiederverlassen schon explizit gelöscht werden. Sie müssen jedoch aufgrund der damit verbundenen Speicherfreigabe als verloren angesehen werden.

Die Befehle LINK und UNLK stellen automatisch die Adressierung der lokalen Daten und Parameter sicher.

Der Befehl LINK kennt zwei verschiedene Parameter:

- das Adreßregister
- die unmittelbare Konstante

Das Adreßregister hat die Aufgabenstellung eines Datenbereichszeigers (FP).

Die Konstante gibt die im Stapel zu speichernde Byteanzahl an.

Der Stapelzeiger wird so gesetzt, daß ein entsprechender Freibereich geschaffen wird. Ferner wird der vorhergehende Wert des Adreßregisters FP wiederhergestellt.

UNLK kehrt den ganzen Vorgang dadurch wieder um, daß Speicherraum freigegeben wird und der alte Inhalt des Zeigers wiederhergestellt wird.

Auf diese Weise sorgt ein LINK-Befehl bei Unterprogrammbeginn für das Laden der Daten in den zugehörigen Speicherbereich und ermöglicht ihre Adressierung mittels des Rahmenzeigers FP.

Ein entsprechender UNLK-Befehl bei Unterprogrammende stellt den alten Stapelzustand vor dem Aufruf wieder her und holt den alten Wert des Speicherbereichszeigers wieder zurück. Diese beiden Befehle gestatten also problemlos, ineinander verschachtelte Unterprogrammaufrufe zu realisieren.

Es ist ebenfalls möglich, rekursive Prozeduraufrufe zu programmieren, da bei jedem erneuten Prozeduraufruf jeweils ein neuer zugehöriger Satz mit lokalen Variablen zur Verfügung gestellt wird.

Übersetzung eines Pascal-Programms in 68000-Assembler:

```
VAR PARAM1, PARAM2: INTEGER;
PROCEDURE PROC (X: INTEGER; VAR Y: INTEGER);
        VAR A, B: INTEGER;
```

```
        BEGIN
        <Abarbeitung der Prozedur>
        END;
BEGIN
        PROC(PARAM1, PARAM2)
END;
```

Dies führt zu:

```
        MOVE    PARAM1 in → -(SP) ;Retten von PARAM1 auf
                                  ;den Stapel
        PEA     PARAM2            ;Retten der Adresse von
                                  ;PARAM2
        JSR     PROC              ;Aufruf von PROC
        ADD     #6 zu SP          ;Setzen des Stapelzeigers auf
                                  ;die ursprüngliche Position
PROC
        LINK    FP, 4             ;Link-Operation
        MOVEM   <Registerliste>   in → -(SP)
                                  ;Retten der Register zu
                                  ;Beginn des Unterprogramms
        <Abarbeitung der Prozedur>
        MOVEM   <Registerliste>   von (SP) +
                                  ;Wiederherstellung der Re-
                                  ;gister vor dem Rücksprung
        UNLK    FP
        RTS
```

Die Verwaltung des Stapels wird durch Abb. 5.30 verdeutlicht.

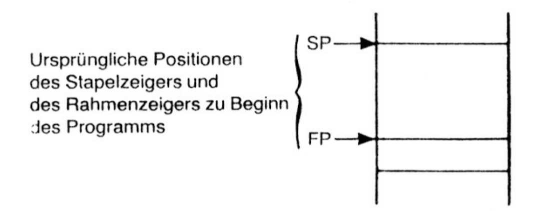
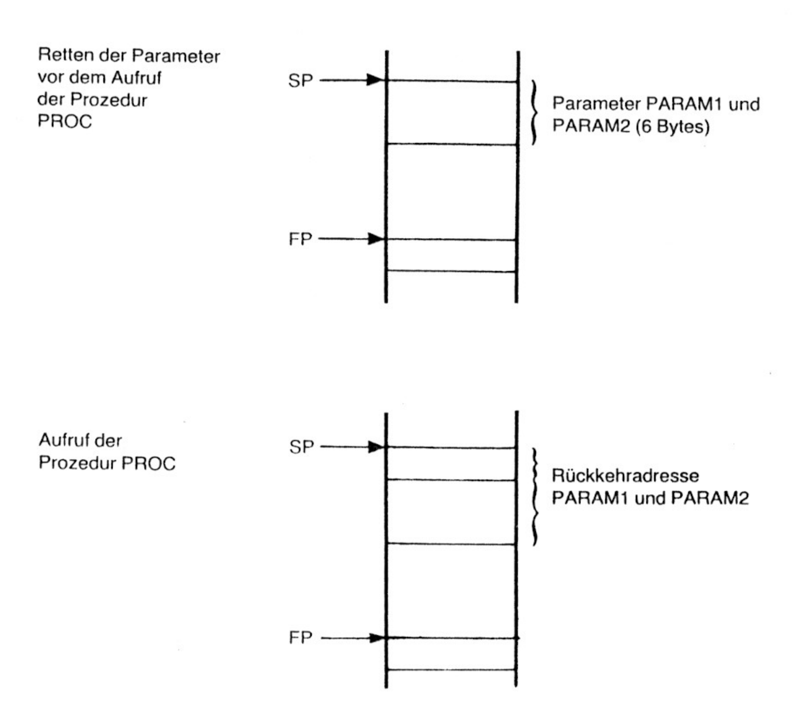


Abb. 5.30: Stapelverwaltung (Umwandlung des Pascal-Programms in Assembler)



Ausführung von LINK

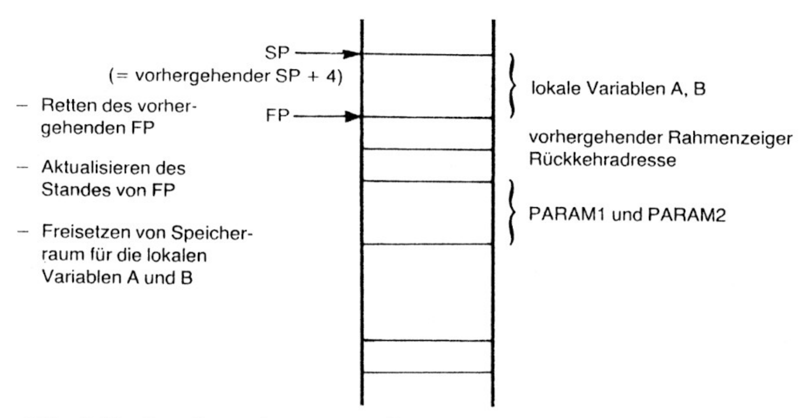
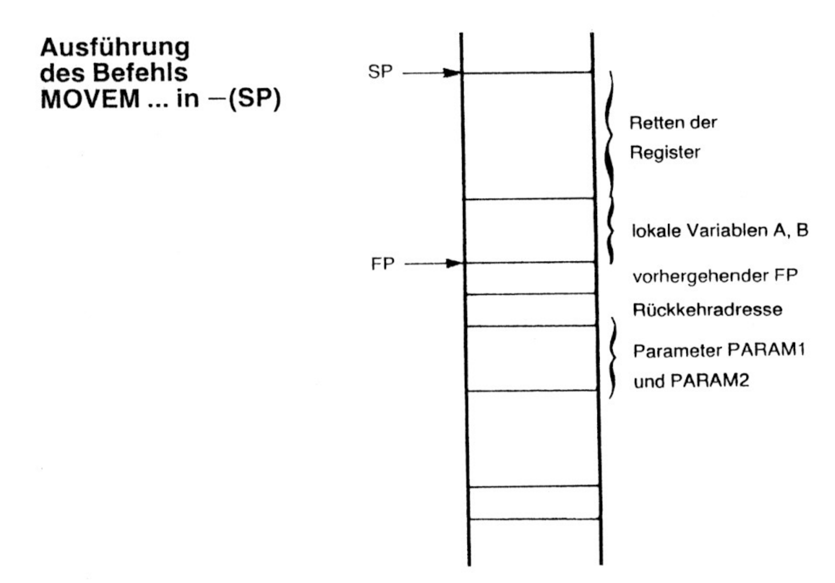


Abb. 5.30: Stapelverwaltung (Umwandlung des Pascal-Programms in Assembler) (Forts.)



Ausführung des Befehls MOVEM ... von (SP) +
nach der Abarbeitung der Prozedur

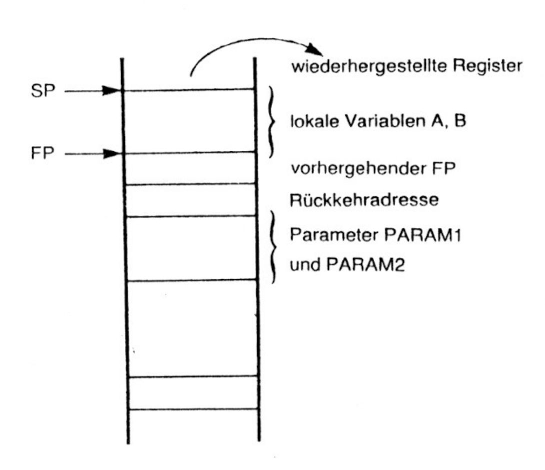


Abb. 5.30: Stapelverwaltung (Umwandlung des Pascal-Programms in Assembler) (Forts.)

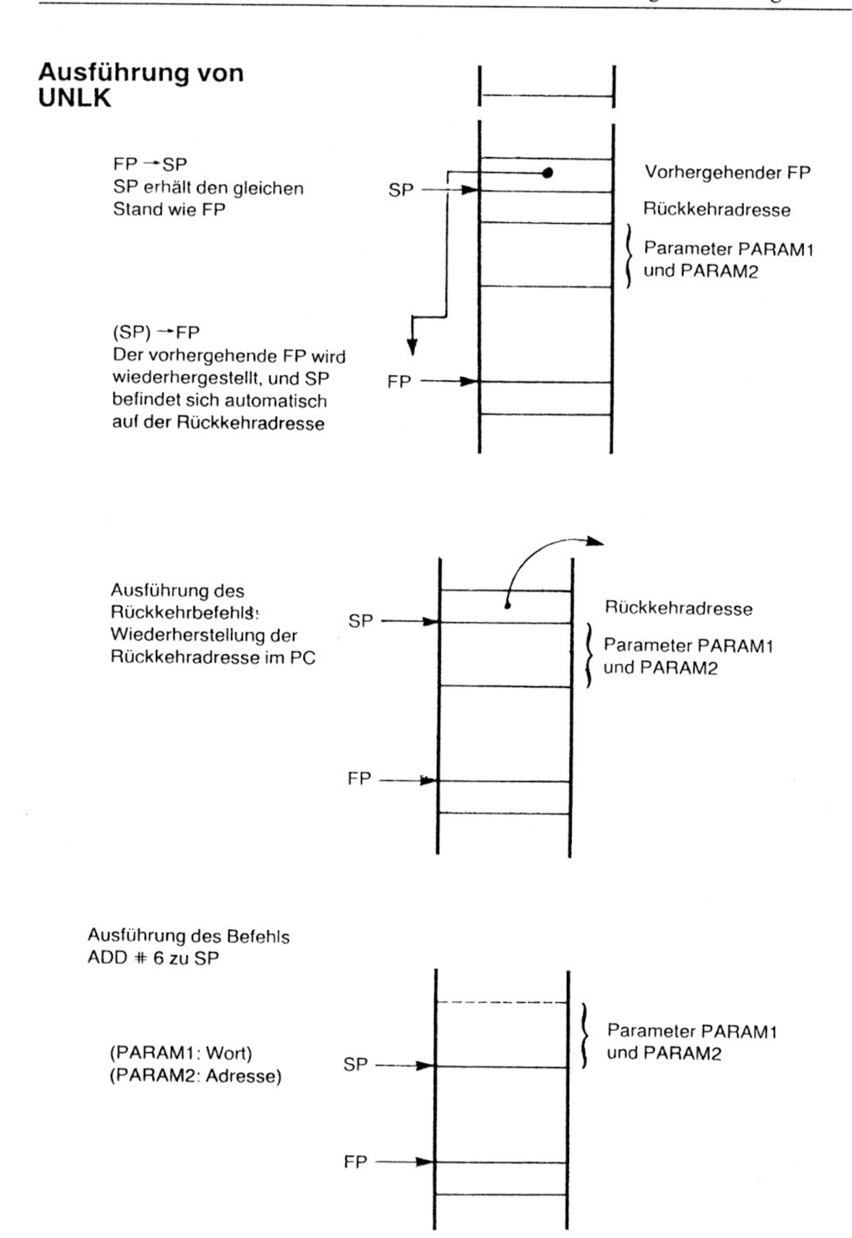


Abb. 5.30: Stapelverwaltung (Umwandlung des Pascal-Programms in Assembler) (Forts.)

Beispiel für die Verwendung von LINK und UNLK in einem Programm:

```
3000           LEA     $2000, A6
3004           LEA     $1FF0, SP
3008  PPROCA
3008           NOP
300A           NOP
300C           NOP
300E           PEA     - 6 (A6)
3012           JSR     PROCB
3016           LEA     4 (SP), SP
301A           NOP
301C           NOP
301E           NOP
3020           ENDA
3022  PROCB
3022           LINK    A6, - $10
3026           NOP
3028           NOP
302A           NOP
302C           UNLK    A6
302E           RTS
               END
```

Die Verwaltung des Stapels wird durch Abb. 5.31 verdeutlicht.

LINK-Aktion

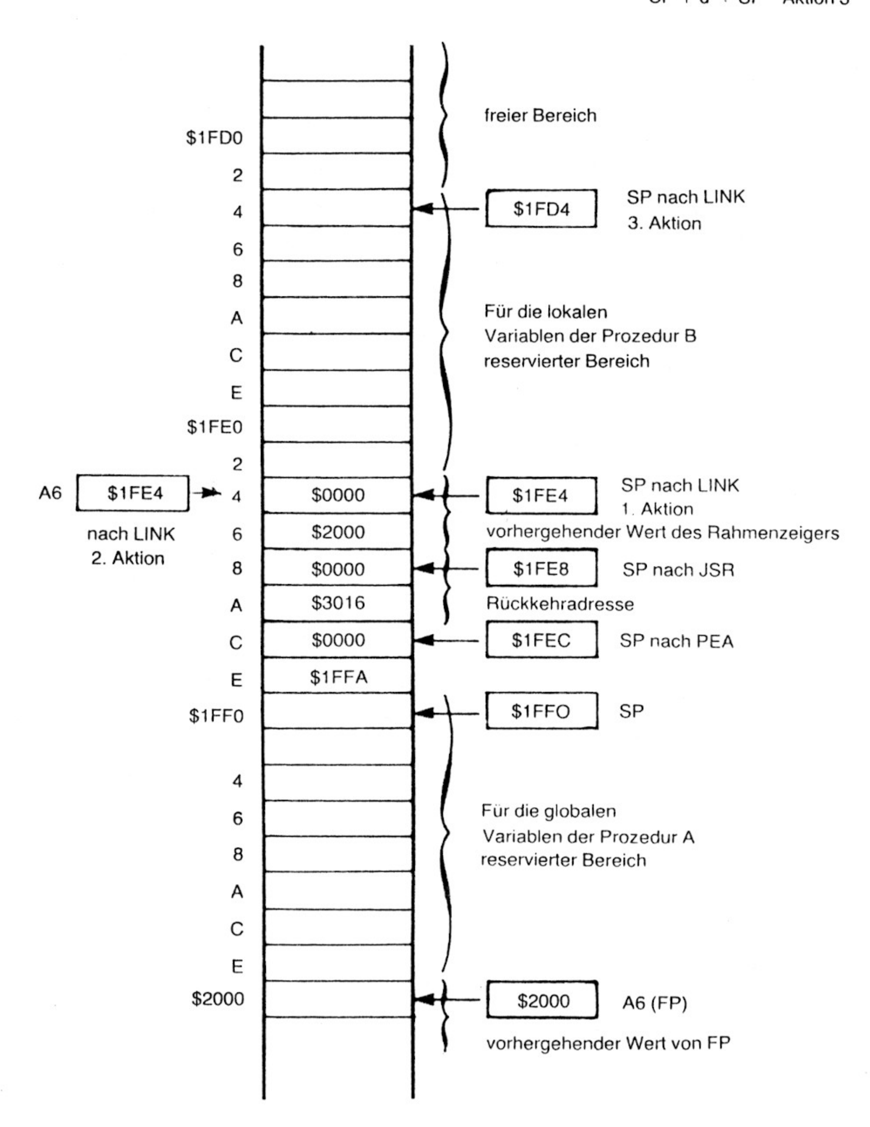


Abb. 5.31: Arbeitsweise von LINK und UNLK

UNLK-Aktion

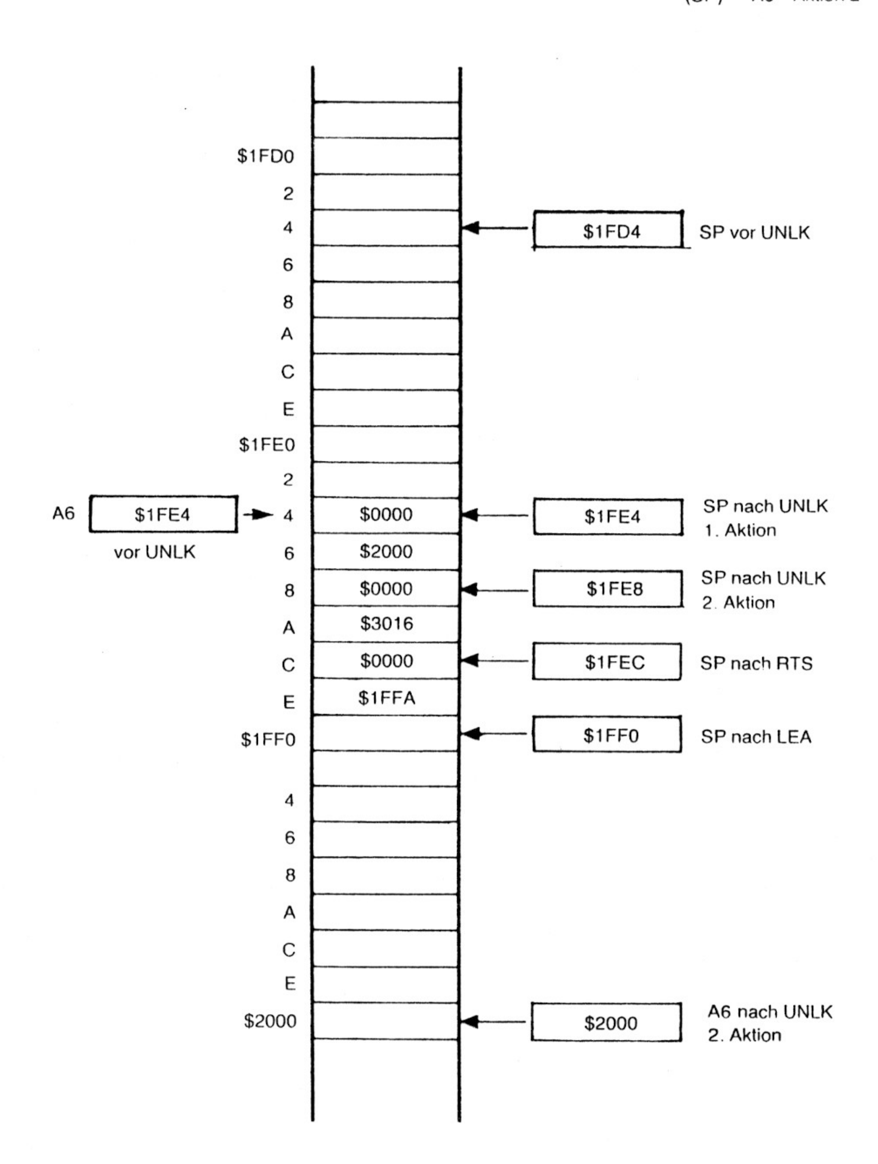


Abb. 5.31: Arbeitsweise von LINK und UNLK (Forts.)

Übungen

5.1: Wie lang ist der Adreßdistanzwert bei den Befehlen Bcc? Wo befindet er sich in bezug auf den Operationscode?

5.2: Wie oft kann D0 beim Befehl DBcc D0,D16 mit (D0)=6 heruntergezählt werden?

5.3: Welches Bit des Bedingungscoderegisters wird bei Vergleichsbefehlen nicht beeinflußt?

5.4: Was bewirkt der Befehl EXT.W D0, wenn D0=$FFEE3210 ist?

5.5: Welche Zieloperanden sind bei den Befehlen DIVS und DIVU möglich?

5.6: Das folgende Programm ruft einen Ausnahmezustand hervor. Welchen und warum?

```
          MOVE.W    # $2000, A0
          MOVEQ.L   # 4, D0
SCHLEIFE
          CMP.W     0(A0, D0) D1
          DBEQ      D0, SCHLEIFE
          END
```

5.7: Welche Befehle in der folgenden Befehlssequenz sind falsch und warum?

```
          ROL.B     # 4, D1
          ROXL.L    D1,D0
          LSL       # 3, (A1)
          ASR.W     D0, D1
          ALS.B     (A0)
          LSR.L     # 3, (A1)
```

Lösungen

5.1: Der Adreßdistanzwert ist 8 Bits lang und befindet sich im Befehlswort selbst, oder er ist 16 Bits lang und ist im Erweiterungswort enthalten.

5.2: D0 kann von 6 bis −1, also 7mal heruntergezählt werden.

5.3: Nur das Bit X wird nicht beeinflußt.

5.4: Nach der Ausführung des Befehls ist D0=$FFEE0010.

5.5: Der Zieloperand kann nur ein Datenregister sein.

5.6: Der Ausnahmezustand wird durch eine illegale Adresse hervorgerufen, denn D0 spielt die Rolle des Schleifenzählers und Indexregisters, so daß D0 beim zweiten Schleifendurchgang den Wert 3 hat, und man muß auf eine ungerade Adresse zugreifen.

Richtig wäre das folgende Beispiel:

```
         MOVE.W    #$2000, A0
         MOVEQ.L   #4, D0
         LEA       2(A0, D0), A0
SCHLEIFE
         CMP.W     -(A0),D1
         DBEQ      D0, SCHLEIFE
         END
```

5.7:

```
ROL.B     # 4, D1      ;korrekt
ROXL.L    D1, D0       ;korrekt
LSL       # 3, (A1)    ;bei Speicherfeldern ist nur
                       ;eine Verschiebung um 1 Bit möglich
ASR.W     D0, D1       ;korrekt
ASL.B     (A0)         ;bei einem Speicherfeld muß der Operand
                       ;ein Wort sein
LSR.L     # 3, (A1)    ;3 und .L sind nicht korrekt, da ein
                       ;Speicherfeld verschoben wird
```

Kapitel 6

Anwenderprogramme

PROGRAMM 1

Funktionsbereich des Bits X:

Das Bit X wird nicht bei allen Befehlen beeinflußt. Das bietet dem Anwender vielfältige Möglichkeiten.

Wird beispielsweise eine Rechnung mit mehrfacher Genauigkeit durchgeführt, bei der ein Übertrag zugespeichert wird, ist es möglich die Bedingungscodes zu verändern, insbesondere den für den Übertrag C, ohne das Bit X zu beeinflussen.

Wir wollen nun eine Reihe Worte addieren. Die erste Reihe ist über Register A0, die zweite über A1 adressierbar. Beide enthalten gleich viele Elemente, und das Register A2 enthält die Endadresse der Reihe.

```
         AND       #$EF, CCR      ;Bit X auf 1 setzen
SCHLEIFE
         ADDX.W    -(A0), -(A1)   ;Addition zweier Worte mit X
                                  ;Wenn ein Übertrag entsteht,
                                  ;werden C und X gesetzt
         CMP.L     A2, A1         ;Test auf Ende der Liste.
                                  ;Dieser Befehl kann C ver-
                                  ;ändern, aber nicht X.
         BLS       SCHLEIFE       ;Wenn das Ende der Liste noch
                                  ;nicht erreicht ist, wird die
                                  ;Schleife erneut durchlaufen
         END
```

PROGRAMM 2

Es soll eine Tabelle von 256 Bytes, die in Doppelworten eingeteilt ist, auf 0 gesetzt werden. Die Startadresse ist MEM (Abb. 6.1).

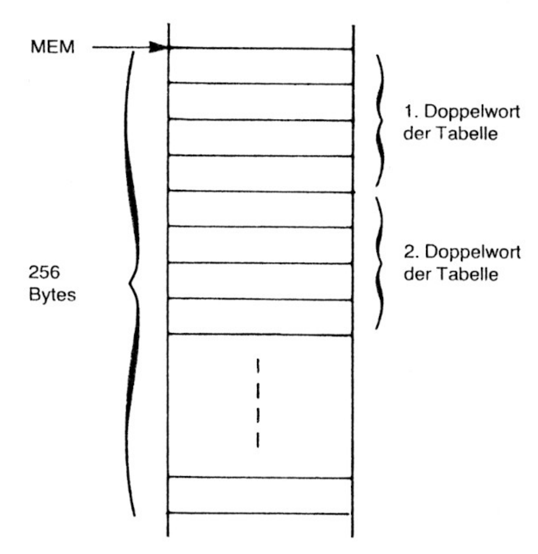


Abb. 6.1: Speicherorganisation (Programm 2)

Das zugehörige Flußdiagramm ist in Abb. 6.2 zu sehen.

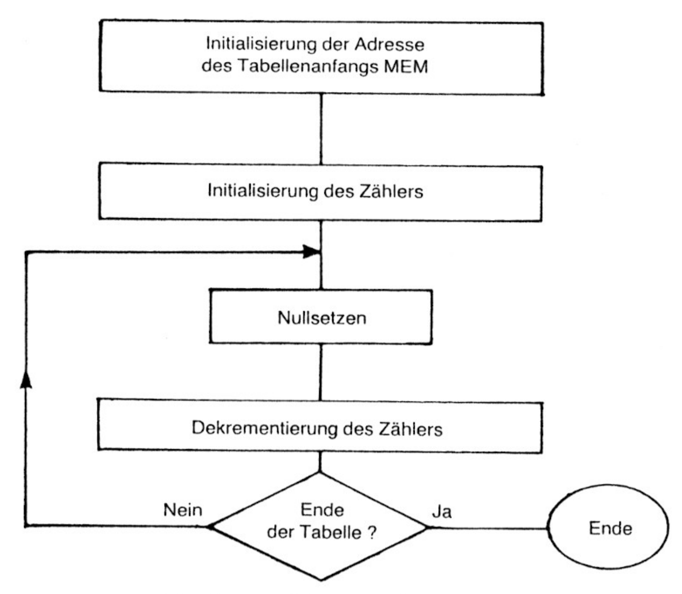


Abb. 6.2: Flußdiagamm (Programm 2)

Das Programm sieht nun folgendermaßen aus:

```
        LEA     MEM, A0          ;Laden von MEM in A0
        MOVE.W  #($100 / 4) - 1, D0
                                 ;Laden des Zählers D0 mit
                                 ;dem Wert 63
LOESCH
        CLR.L   (A0) +           ;Nullsetzen des durch A0
                                 ;adressierten Doppelwortes
                                 ;und Erhöhung von A0 um 4
        DBRA    D0, LOESCH       ;Dekrementieren von D0.
                                 ;Wenn D0 ungleich -1 ist, geht
                                 ;es zurück zu RAZ, wenn D0
                                 ;gleich -1 ist, dann ist das
                                 ;Ende erreicht
        END
```

PROGRAMM 3

In den folgenden drei Beispielen wird ein Speicherbereich auf 0 gesetzt, der durch A0 und A1 begrenzt ist (A1>=A0).

1. Fall: Für A0 und A1 werden keine Bedingungen gestellt.
2. Fall: A0 und A1 enthalten gerade Adressen.
3. Fall: A0 und A1 enthalten gerade Adressen, die durch 4 teilbar sind.

Programm zum 1. Fall:

```
         CLR.B    D0             ;D0 = x x x x x x 0 0
SCHLEIFE
         MOVE.B   D0, (A0) +     ;(A0) = 00 und Inkrementie-
                                 ;rung von A0
         CMPA.L   A0, A1         ;Test auf Ende des Speicher-
                                 ;feldes
         BPL      SCHLEIFE       ;Solange A0 < A1 ist, wird die
                                 ;Schleife durchlaufen
         END
```

Programm zum 2. Fall:

```
         CLR.W    D0
SCHLEIFE
         MOVE.W   D0, (A0) +
```

```
        CMPA.L  A0, A1
        BPL     SCHLEIFE
        END
```

Programm zum 3. Fall:

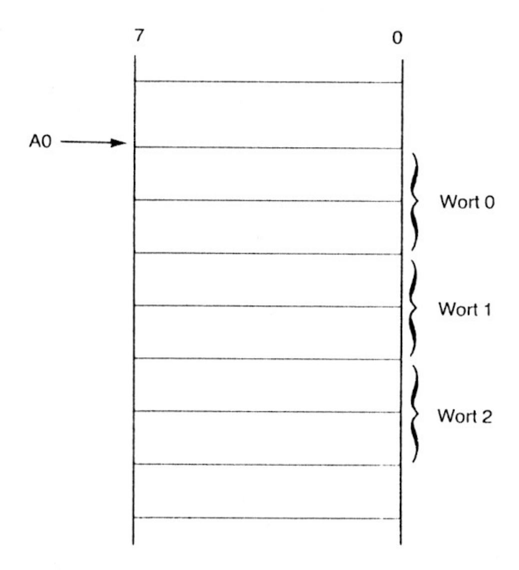
```
SCHLEIFE
        CLR.L   (A0) +
        CMPA.L  A1, A0
        BPL     SCHLEIFE
        END
```

PROGRAMM 4

Die Addition von 10 Datenworten soll realisiert werden:

Wort 0 + Wort 1 + ... + Wort 9

Im Speicher soll es folgendermaßen aussehen:



Das Ergebnis soll in Register D0 abgelegt werden, während das Adreßregister A0 auf das erste zu addierende Wort zeigt. Es müssen Zahlen mit und ohne Vorzeichen berücksichtigt werden.

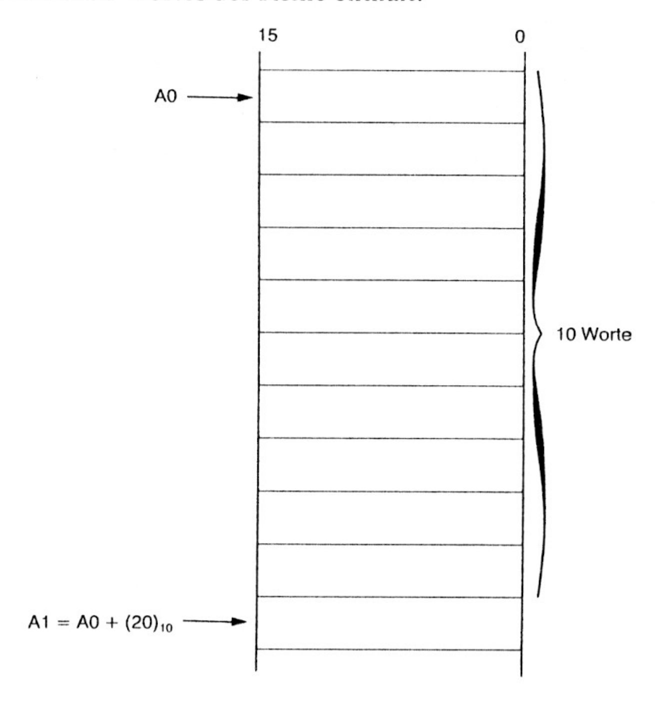
Zahlen ohne Vorzeichen:

1. Lösung:

```
            CLR.L     D0            ;D0 = 00000000
            CLR.L     D1            ;D1 = 00000000
            MOVE.L    #$9, D2       ;D2 = 00000000
                                    ;D2: Schleifenzähler
START
            MOVE.W    (A0) +, D1    ;Laden des zu addierenden
                                    ;Wortes in D1
            ADD.L     D1, D0        ;Addition des Wortes in D1 zu
                                    ;der bereits aufgelaufenen
                                    ;Summe in D0
            DBRA      D2, START     ;Wenn D2 = -1, dann Ende,
                                    ;sonst erneuter Schleifen-
                                    ;durchlauf
            END
```

2. Lösung:

Der Wert von A1 wird berechnet, der die Adresse des ersten nicht mehr zu addierenden Wortes der Reihe enthält.



```
         MOVE.L   A0, A1          ;Kopieren der Startadresse
                                  ;nach A1
         ADD.L    #$14, A1        ;A1 = A0 + 20 (Bytes)
         CLR.L    D0              ;D0 = 00000000
         CLR.L    D1              ;D1 = 00000000
SCHLEIFE
         MOVE.W   (A0) +, D1      ;Laden des zu addierenden
                                  ;Wortes in D1
         ADD.L    D1, D0          ;Addieren des Wortes zu der
                                  ;bereits aufgelaufenen Summe
         CMPA.L   A1, A0          ;Vergleich zwischen aktueller
                                  ;Adresse und Endadresse
                                  ;der Tabelle
         BNE      SCHLEIFE        ;Verzweigung nach SCHLEIFE,
                                  ;wenn Z = 0, also A0 # A1;
                                  ;sonst Ende
         END
```

Zahlen mit Vorzeichen:

Zur Abwechslung wird diesmal der Befehl DBcc nicht verwendet.

1. Lösung:

```
         CLR.L    D0              ;D0 = 00000000
         MOVE.B   #9, D2          ;D2 = xxxxxx09
                                  ;D2 Schleifenzähler
SCHLEIFE
         MOVE.W   (A0) +, D1      ;Speicherinhalt in D1 laden
         EXT.L    D1              ;Vorzeichenerweiterung
         ADD.L    D1, D0          ;(D1) + (D0) in D0
         SUB.B    #1, D2          ;(D2) = (D2) - 1
         BCC      SCHLEIFE        ;Für D2 = xxxxxxFF, C = 1
         END
```

2. Lösung:

```
         CLR.L    D0              ; D0 = 00000000
         MOVE.B   #$0A, D2        ;D2 = xxxxxx0A
SCHLEIFE
         MOVE.W   (A0) +, D1      ;Speicherinhalt in D1
         EXT.L    D1              ;Vorzeichenerweiterung
         ADD.L    D1, D0          ;(D1) + (D0) in D0
         SUB.B    #1, D2          ;(D2) = (D2) - 1
         BNE      SCHLEIFE        ; Wenn D2 = xxxxxx00, Z = 1
         END
```

PROGRAMM 5

Es wird der größte Wert in einer Tabelle aus 10 Worten ohne Vorzeichen gesucht.

A0 enthält die Anfangsadresse der Tabelle, und D0 nimmt das Ergebnis auf.

Das Flußdiagramm des Programms sehen Sie in Abb. 6.3.

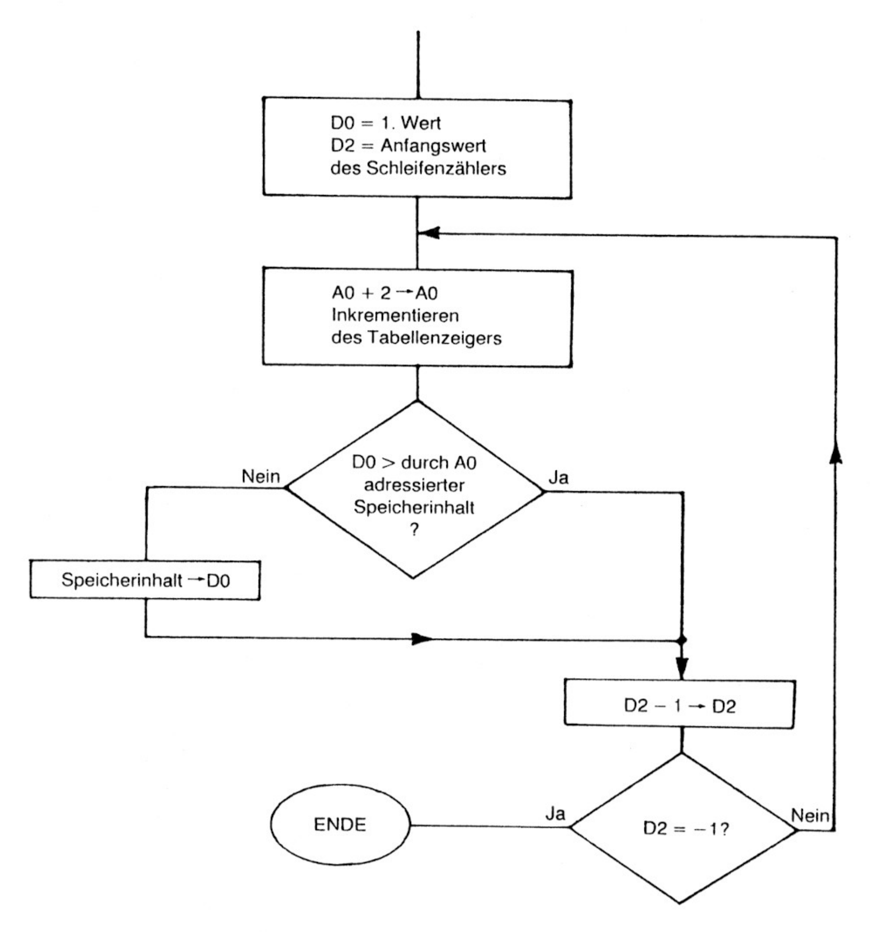


Abb. 6.3: Flußdiagramm (Programm 5)

Das Programm lautet folgendermaßen:

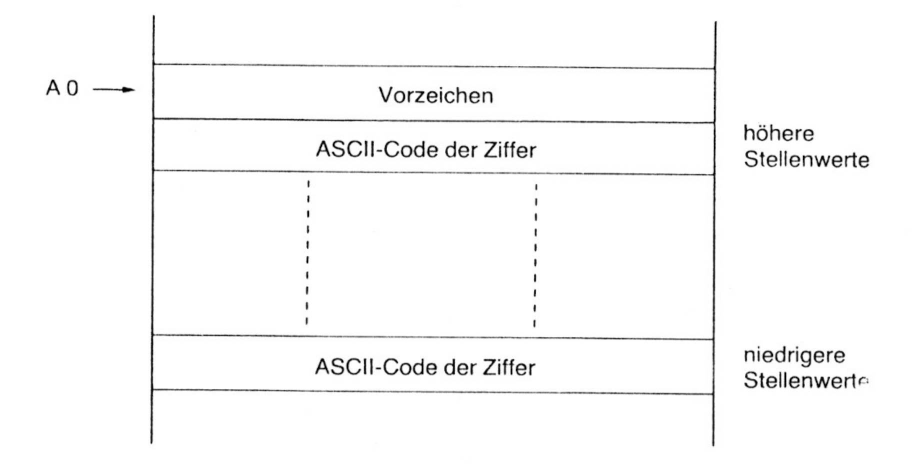
```
         MOVEQ    #9,D2           ;D2 : Schleifenzähler
         MOVEQ    #2,D1           ;D1 : Indexregister
         MOVE.W   (A0),D0         ;Ersten Wert in D0 laden
SCHLEIFE
         CMP.W    0(A0,D1),D0     ;Vergleich zwischen dem
                                  ;aktuellen Wert und dem bis-
                                  ;herigen Maximum
         BPL      POITIV          ;N = 1 wenn D0 < (A0,D1)
         MOVE.W   0(A0,D1),D0     ;Neues Maximum in D0
                                  ;schreiben
POSITIV
         DBRA     D2,SCHLEIFE     ;Ende wenn D2 = -1
         END
```

PROGRAMM 6

Wir wandeln eine Dezimalzahl mit Vorzeichen, die binär in 32 Bits verschlüsselt ist, in ASCII-Code um.

Die zu konvertierende Zahl steht in Register D0.

Das Ergebnis soll im Speicher ab Adresse A0 stehen, und zwar in folgender Form:



Die umzuwandelnde Zahl soll aus maximal 10 Dezimalziffern bestehen. Folglich werden für das Ergebnis 11 Bytes benötigt, 10 für die Zahl und 1 für das Vorzeichen.

Wir gehen von der folgenden Tabelle aus:

TABWERT	1000000000	Wert 1
	100000000	Wert 2
	10000000	Wert 3
	1000000	
	100000	
	10000	
	1000	
	100	
	10	
	1	

Es wird nun der erste Wert dieser Tabelle von der umzuwandelnden Zahl subtrahiert und nachher überprüft, ob diese Zahl kleiner war als der Tabellenwert. Die Anzahl der Subtraktionen, die durchgeführt werden müssen, bis ein negatives Ergebnis auftritt, ist dann die in ASCII zu verschlüsselnde Zahl.

Um die nächste Zahl zu berechnen, werden die Subtraktionen mit dem Wert, der sich ergeben hatte, bevor das negative Ergebnis auftrat, und dem nächsten Tabellenwert wiederholt.

Beispiel:

```
 2xxxxxxxxx   Dezimalziffern
-1000000000
-----------
 1xxxxxxxxx
```

Das Ergebnis ist positiv, also weiter:

```
 1xxxxxxxxx
-1000000000
-----------
 0xxxxxxxxx
```

Das Ergebnis ist immer noch positiv:

```
 0xxxxxxxxx
-1000000000
-----------
```

negatives Ergebnis, also Stop.

Es waren also zwei Subtraktionen nötig, bis ein negatives Ergebnis auftrat. Die Zahl, die codiert werden soll, ist demnach 2.

Bemerkung: Das Ergebnis 0xxxxxxxxx muß gesichert werden, damit die Berechnung weitergehen kann. In unserem Beispiel muß dazu der Wert 1000000000 zu dem negativen Ergebnis addiert werden.

ASCII-Tabelle der Dezimalziffern

dezimal	ASCII (hex)
0	30
1	31
2	32
3	33
4	34
5	35
6	36
7	37
8	38
9	39

Mit dem Programm finden wir sofort den ASCII-Code der Ziffer, und zwar abhängig von der Anzahl der Subtraktionen. Bei jeder Subtraktion wird – solange das Ergebnis größer als 0 ist – die Zahl $D0 dekrementiert, die das Zweierkomplement der Zahl $30 ist.

Auf diese Weise wird bei der Zahl 1 eine einzige Subtraktion durchgeführt, nämlich:

$D0−1=$CF

Im Zweierkomplement ergibt das $31, den ASCII-Code von 1.

Für die Zahl 2 sind zwei Subtraktionen notwendig, nämlich noch:

$CF−1=$CE

Im Zweierkomplement ist das $32, der ASCII-Code von 2.

Das Flußdiagramm dieses Programms zeigen wir in Abb. 6.4.

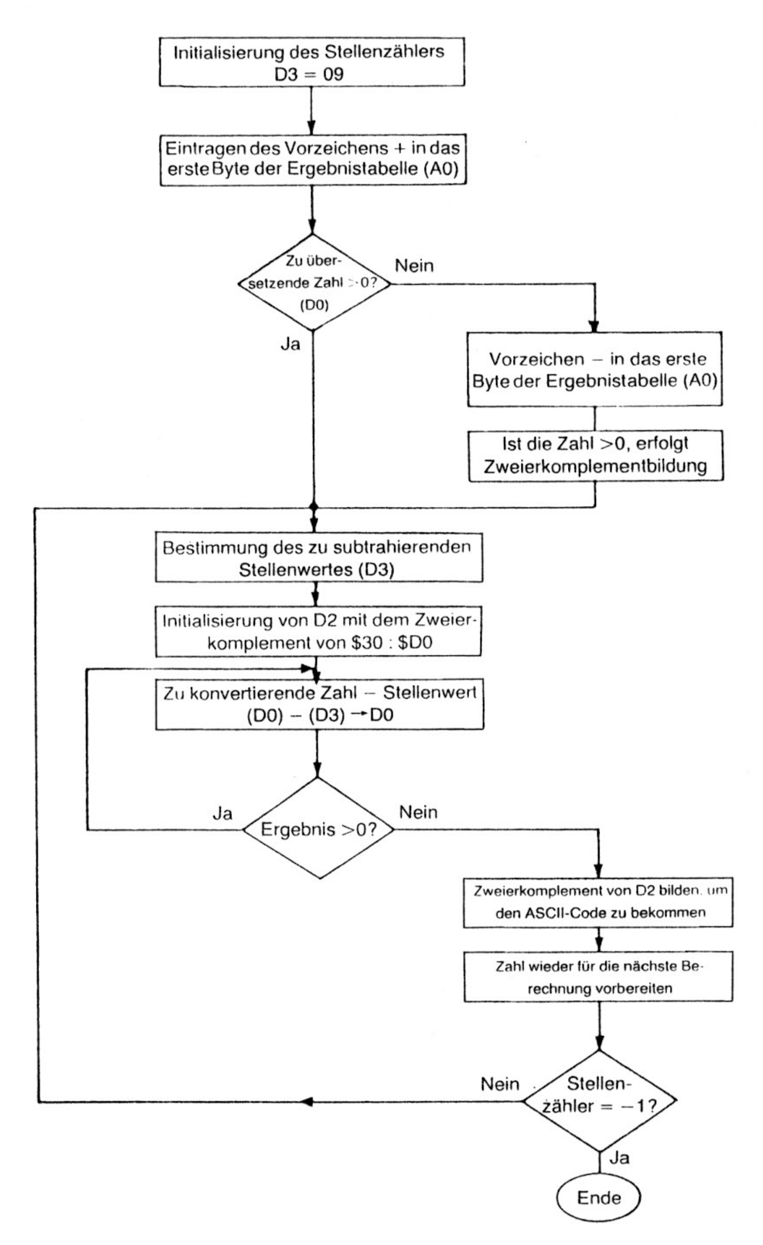


Abb. 6.4: Flußdiagramm (Programm 6)

Das Programm dazu sieht so aus:

```
         MOVEQ    #9, D1           ;Initialisierung des Stellenzählers
         MOVE.B   #$2B, (A0)       ;ASCII-Code des Vorzeichens +
                                   ;in die Ergebnistabelle eintragen
         TST.L    D0               ;Test, ob die zu konvertierende
                                   ;Zahl positiv oder negativ ist
         BGE      POSITIV          ;Wenn positiv, Fortsetzung des
                                   ;Programms
         MOVE.B   #$2D, (A0)       ;ASCII-Code des Vorzeichens −
                                   ;in die Ergebnistabelle eintragen
         NEG.L    D0               ;Zweierkomplement der Zahl
                                   ;bilden
POSITIV
         ADDQ     #1, A0           ;Vorbereiten von A0 für das
                                   ;Laden des nächsten Ergebnis-
                                   ;bytes in Speicher
         LEA      TABWERT, A1      ;A1 = Adresse von TABWERT
WERT
         MOVE.L   (A1)+, D3        ;Laden des Stellenwertes in D3
         MOVEQ    #$D0, D2         ;D2 = $30
SUBTR
         SUB.L    D3, D0           ;Subtraktion des Stellenwertes
         DBMI     D2, SUBTR        ;Wenn Ergebnis > 0, muß eine
                                   ;weitere Subtraktion durchge-
                                   ;führt werden, und D2 muß
                                   ;dekrementiert werden
                                   ;Wenn Ergebnis < 0, erfolgt
                                   ;keine weitere Subtraktion, und
                                   ;D2 bleibt unverändert
         NEG.B    D2               ;Übergang zur positiven ASCII-
                                   ;Darstellung
         MOVE.B   D2, (A0)+        ;Eintragen in die Ergebnistabelle
         ADD.L    D3, D0           ;Es wurde eine Subtraktion
                                   ;zuviel ausgeführt
         DBRA     D1, WERT         ;Schon 10 Ziffern durch:
                                   ;Ja, dann Ende
                                   ;Nein: Der Zyklus der Subtrak-
                                   ;tion beginnt von vorn
         END
```

PROGRAMM 7

Wir wollen eine 64-Bit-Addition mit den Zahlen N1 und N2 durchführen, die, wie in Abb. 6.5 gezeigt, im Speicher liegen.

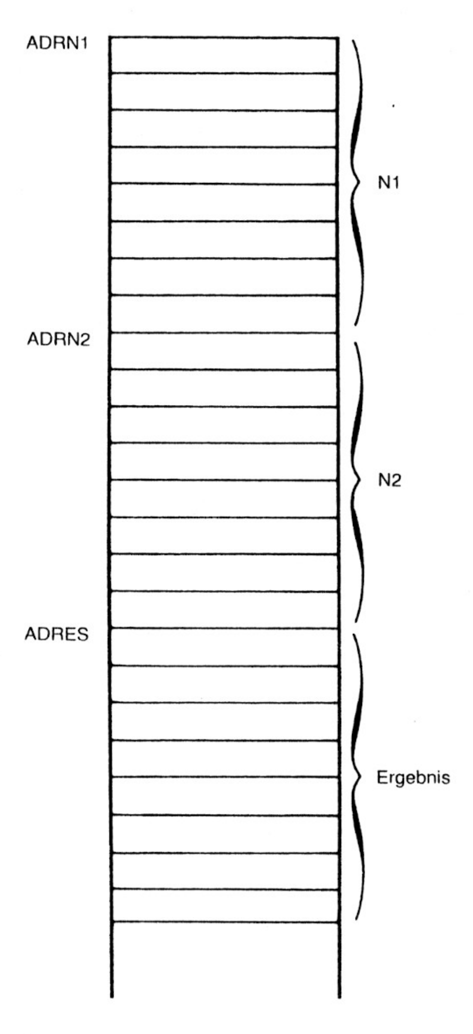


Abb. 6.5: Speicherorganisation (Programm 7)

Um diese 64-Bit-Addition zu realisieren, führen wir zwei 32-Bit-Additionen mit Hilfe des ADDX-Befehls aus, der uns auch das Setzen der Bedingungscodes X und Z ermöglicht:

- Das Erweiterungsbit X muß mit 0 initialisiert werden.
- Der Bedingungscode Z muß mit 1 initialisiert werden, denn der ADDX-Befehl bewirkt ein Zurücksetzen von Z auf 0, wenn das Ergebnis nicht 0 ist, sonst bleibt Z unverändert.

Beispiel:

Z=1 zu Beginn (vor ADDX)

1. Fall:

- Wenn das erste Ergebnis 0 ist, bleibt Z unverändert, also 1.
- Wenn das zweite Ergebnis nicht 0 ist, wird Z auf 0 gesetzt.
- Das Endergebnis ist also ungleich 0 und Z=0, was korrekt ist.

2. Fall:

- Wenn das erste Ergebnis nicht 0 ist, wird Z auf 0 gesetzt.
- Wenn das zweite Ergebnis 0 ist, bleibt Z unverändert, also 0.
- Das Endergebnis ist also ungleich 0 und Z=0, was korrekt ist.

3. Fall:

- Wenn das erste Ergebnis 0 ist, bleibt Z unverändert, also 1.
- Wenn das zweite Ergebnis 0 ist, bleibt Z unverändert, also 1.
- Das Endergebnis ist also 0 und Z=1, was korrekt ist.

Dasselbe Prinzip gilt für die Befehle ABCD, NBCD, NEGX, SBCD und SUBX.

Bemerkung: Um das Ergebnis in den Speicher zu bringen, darf keinesfalls der MOVE-Befehl verwendet werden, da er die Bedingungscodes verändert. Dann würde nämlich im 2. Fall, bei dem das Ergebnis 0 war, durch den MOVE-Befehl der Bedingungscode Z auf 1 gesetzt, was falsch wäre.

Um dieses Problem zu lösen, setzt man den Befehl MOVEM ein. Das Programm läßt sich nunmehr ohne Schwierigkeiten verwirklichen:

```
         MOVEQ    #1,D2           ;Initialisierung des Schleifen-
                                  ;zählers
         LEA      (ADRN1+8),A0    ;Setzen des Zeigers in A0
                                  ;unter Berücksichtigung der
                                  ;Predekrementierung
         LEA      (ADRN2+8),A1    ;gleichfalls für A1
         LEA      (ADRES+8),A2    ;gleichfalls für A2
         MOVE     #4,CCr          ;X = 0, Z = 1
SCHLEIFE
         MOVE.L   -(A0),D0        ;Ersten Operanden in D0
                                  ;laden
         MOVE.L   -(A1),D1        ;Zweiten Operanden in D1
                                  ;laden
         ADDX.L   D0,D1           ;Addition
         MOVEM.L  D1,-(A2)        ;Ergebnis in den Speicher
                                  ;schreiben
         DBRA     D2,SCHLEIFE     ;Wenn D2 = -1, dann Ende,
                                  ;sonst erneuter Schleifen-
                                  ;durchlauf
         END
```

PROGRAMM 8

Wir möchten mit Hilfe eines Unterprogramms Worte einer Tabelle in absteigender Reihenfolge sortieren.

Das Register A0 zeigt auf die Adresse des Tabellenanfangs und wird mit ANFTAB initialisiert.

Die Tabelle endet bei der Adresse ENDTAB.

Vorgehensweise: Nacheinander müssen jeweils zwei Elemente N0 und N1 miteinander verglichen werden:

- Ist N0>N1, dann wird zum nächsten Vergleich mit N1 und N2 übergegangen.
- Ist N0<N1, dann werden die Plätze von N0 und N1 in der Tabelle vertauscht, und die Tabelle wird von vorn durchgearbeitet.

ANFTAB	N0
	N1
	N2
	N3

Beispiel:	
	00
	04
	00
	02
	01
	03

0004 > 0002 – kein Tausch
0002 < 0103 – Werte werden vertauscht

00
04
01
03
00
02

Die Untersuchung beginnt wieder am Anfang der Tabelle:

0004 < 0103 – Werte werden vertauscht

01
03
00
04
00
02

Das Flußdiagramm dieser Problemlösung ist in Abb. 6.6 zu sehen.

Das Programm sieht so aus:

```
        LEA     ENDTAB, A1  ;Initialisierung
        LEA     ANFTAB, A0  ;Initialisierung
        MOVE.L  A0, A2      ;Retten der Anfangsadresse
START
        MOVE.L  A2, A0      ;Wiederherstellen der Anfangs-
                            ;adresse
FOLGE
        CMPM.W  (A0)+,(A0)  ;Vergleich zweier aufeinander-
                            ;folgender Worte
        BHI     TAUSCH      ;Abhängig vom Ergebnis des
                            ;Vergleichs werden die Worte
                            ;getauscht oder nicht
        TST.W   -(A0)       ;Dekrementieren von A0
        CMP.L   A0, A1      ;Test auf Tabellenende
        BNE     FOLGE       ;Wenn Z=0, gehe zu FOLGE
                            ;Rückkehr aus dem Unter-
                            ;programm
TAUSCH
        MOVE.W  -(A0), D0   ;Austausch von 2 Worten in der
                            ;Tabelle
```

```
MOVE.W  -(A0),D1     ;
MOVEM   D0/D1,(A0)   ;
BRA     START        ;
END
```

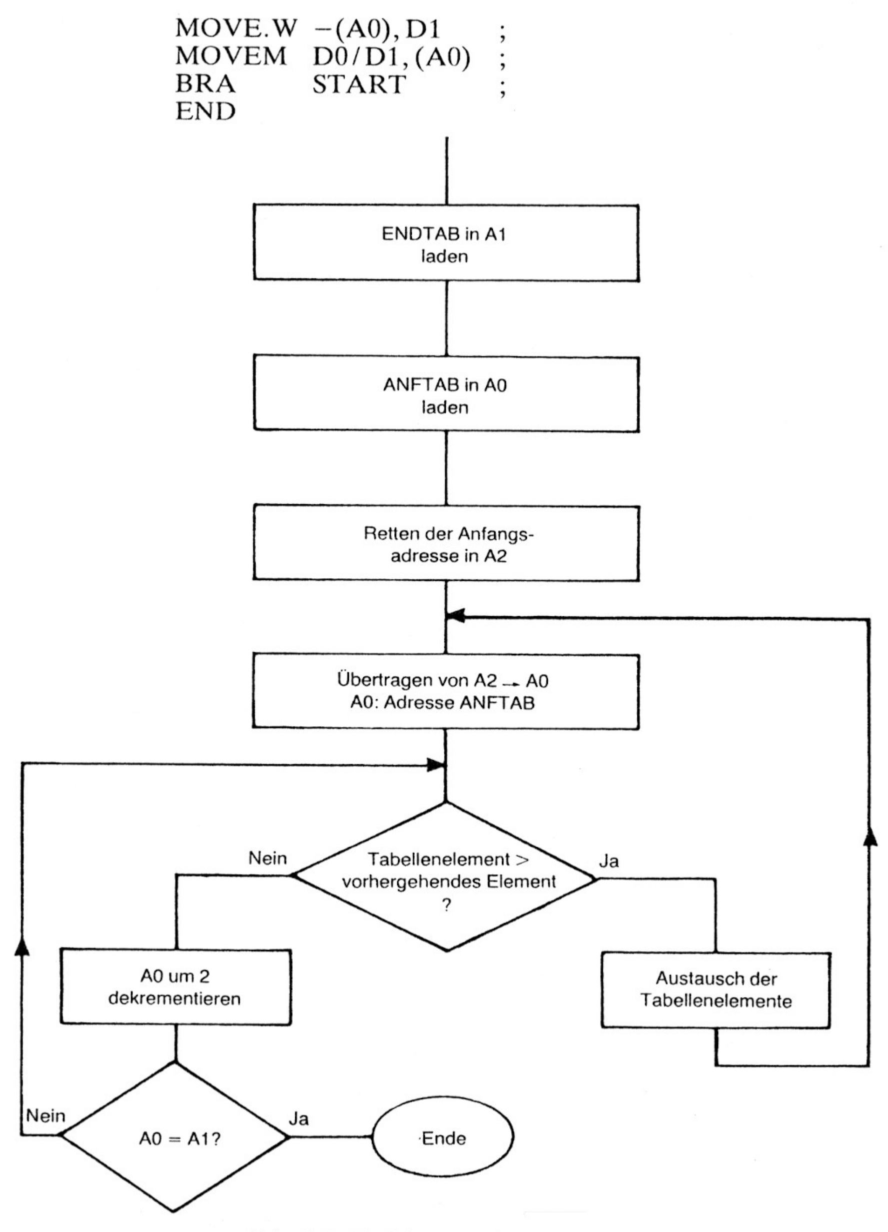


Abb. 6.6: Flußdiagramm (Programm 8)

PROGRAMM 9

Wir wollen eine 32-Bit-Multiplikation zweier Zahlen ohne Vorzeichen durchführen.

Die Operanden N1 und N2 sowie die Zeiger sind im Schema der Abb. 6.7 zu sehen.

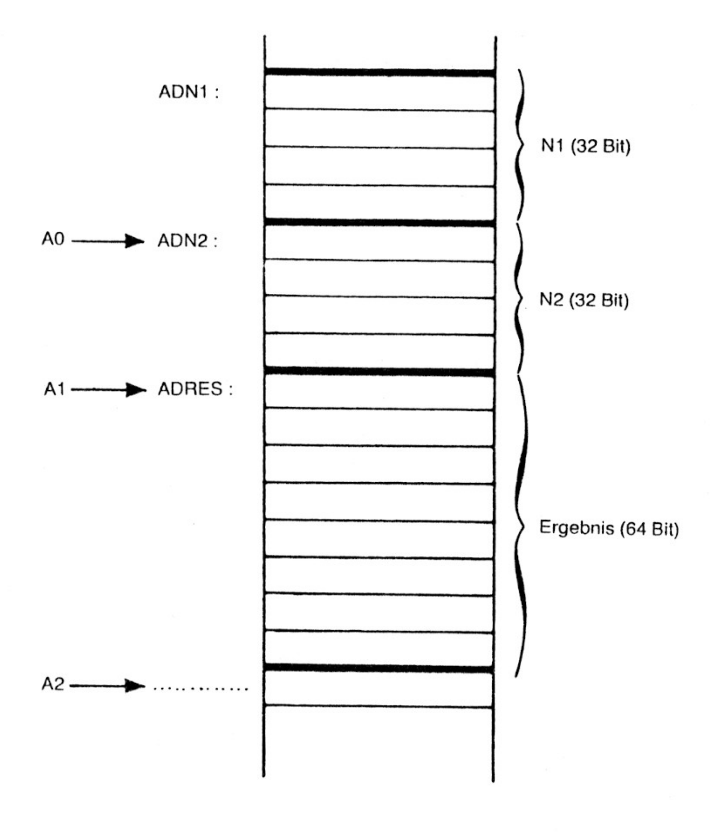


Abb. 6.7: Speicherorganisation (Programm 9)

Das Rechenschema mit N1=33334444 und N2=11112222 ist in Abb. 6.8 veranschaulicht.

N1 = 33334444 et N2 = 1111 2222.

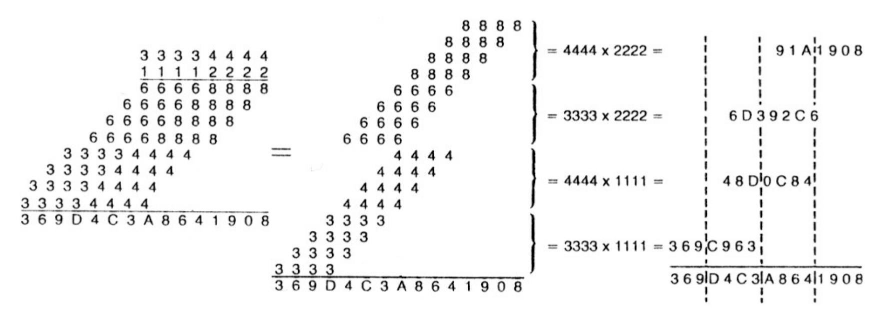


Abb. 6.8: Rechenschema (Programm 9)

In diesem Beispiel wird das Ergebnis im Speicher gebildet, so wie die Abb. 6.9 es zeigt.

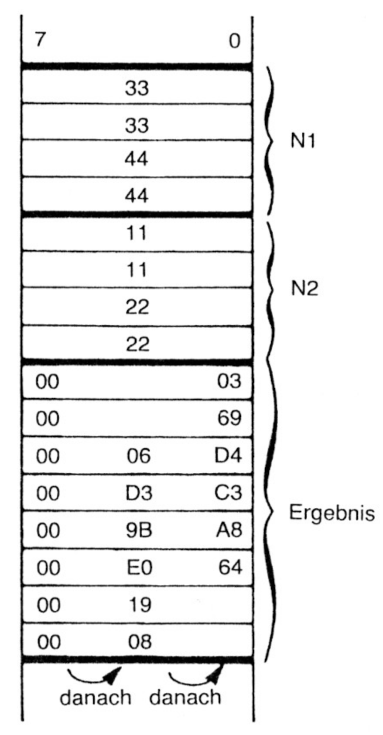


Abb. 6.9: Speicheraufbau nach dem Rechenbeispiel (Programm 9)

Das Programm sieht so aus:

```
LAENGE   EQU       1                       ;Länge der Operanden in
                                           ;Bytes
         LEA       ADN1 + 4, A0            ;A0 = Adresse des nieder-
                                           ;wertigen Anteils von N1 + 1
         LEA       ADN2 + 4, A1            ;gleichfalls für N2
         LEA       ADRES, A2               ;A2 = Adresse des höher-
                                           ;wertigen Anteils des zu-
                                           ;künftigen Ergebnisses
         MOVE.L   (2 * LAENGE) / 2 - 1, D0
                                           ;Ergebnis nullsetzen
LOESCH
         CLR.W     (A2)+                   ;
         DBRA      D0, LOESCH              ;
         MOVE.L   # LAENGE / 2 - 1, D1
                                           ;D1 = 1, um den niederwer-
                                           ;tigen Anteil von N2 mit
                                           ;dem niederwertigen Anteil
                                           ;von N1 zu multiplizieren,
                                           ;danach mit dem höherwer-
                                           ;tigen Anteil von N1
MULT
         CLR.L     D0
         MOVE.L   # LAENGE / 2 - 1, D2
                                           ;D2=1, um den höherwerti-
                                           ;gen Anteil von N2 mit dem
                                           ;niederwertigen Anteil von
                                           ;N1 zu multiplizieren, da-
                                           ;nach mit dem höherwerti-
                                           ;gen Anteil von N1.
         MOVE.W -(A1), D4                  ;Laden der Operanden
SCHLEIFE
         MOVE.W -(A0), D3                  ;
         MULU      D4, D3                  ;Multiplikation
         ADD.L     D3, D0                  ;Addition der vorhergehen-
                                           ;den niederwertigen Anteile
         CLR.L     D3
         MOVE.W -(A2), D3                  ;Addition des vorhergehen-
                                           ;den Ergebnisses und Er-
                                           ;stellen des endgültigen Er-
                                           ;gebnisses.
         ADD.L     D3, D0                  ;
         MOVE.W D0, (A2)                   ;
```

```
CLR.W   D0                    ;Übertragung des höherwer-
                              ;tigen Anteils in den nieder-
                              ;wertigen
SWAP    D0                    ;
DBRA    D2,SCHLEIFE
MOVE    D0,-(A2)              ;Erstellung des Ergebnisses
LEA     LAENGE(A0),A0         ;Reinitialisierung, um einen
                              ;anderen Operandenteil zu
                              ;finden und die vorherge-
                              ;henden Ergebnisse für die
                              ;zukünftigen Additionen
                              ;anzurechnen
LEA     LAENGE(A2), A2        ;
DBRA    D1,MULT
END
```

Kapitel 7

Die anderen Prozessoren der 68000-Familie

DER MC68008

Hier handelt es sich um einen Prozessor aus der 68000-Familie, der zwar intern eine 32-Bit-Architektur hat, aber einen auf 8 Bits reduzierten Datenbus D0–D7 besitzt.

Dieser Mikroprozessor mit seinen 48 Anschlußstiften stellt eine weniger komplexe und kostengünstigere Alternative dar, die trotzdem die Vorteile eines 32-Bit-Systems bietet.

Die interne Struktur des MC68008 ist mit der des MC68000 identisch, d. h. er besitzt 17 32-Bit-Register:

8 Datenregister: D0, ..., D7
7 Adreßregister: A0, ..., A6
1 Stapelzeigerregister: A7
1 Programmzähler: PC

Das 16-Bit-Statusregister SR hat den gleichen Aufbau wie das des 68000.

Auch der Stapelzeiger A7 hängt wie beim 68000 unmittelbar vom Zustand des Prozessors ab (Benutzer- oder Überwachermodus).

Dennoch gibt es gewisse Unterschiede zwischen dem 68008 und dem 68000. Abb. 7.1 zeigt die Anschlüsse des 68008.

Die Unterschiede, die zu beachten sind, sind die folgenden:

- Es gibt nur eine einzige Stromversorgung VCC.
- Bei der Buskontrolle im Synchron-Modus wird nicht das Signal $\overline{VMA}$ erzeugt.

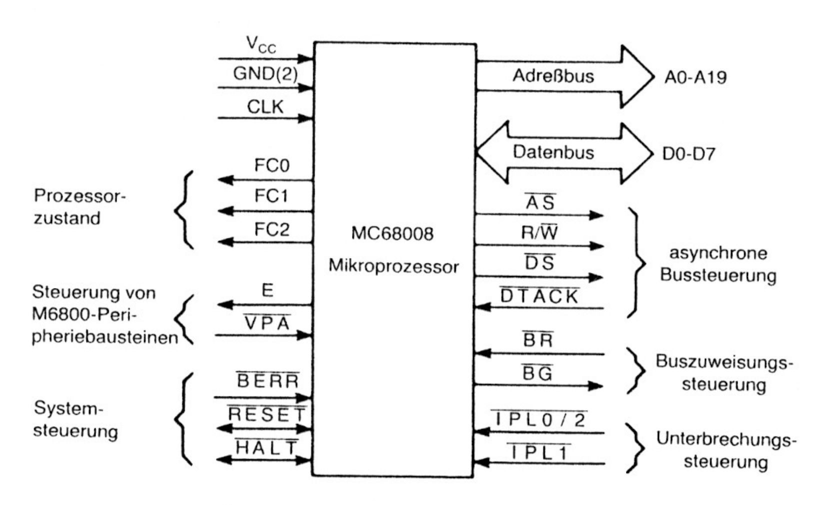


Abb. 7.1: Die Anschlüsse des MC68008

- Der Adreßbus umfaßt die Adreßleitungen A0 bis A19.
- Der Datenbus ist ein 8-Bit-Bus: D0 bis D7.
- Bei der Bussteuerung im asynchronen Modus erfolgt die Bestätigung der Daten über einen $\overline{DS}$-Anschlußstift anstelle von $\overline{UDS}$ und $\overline{LDS}$.
- Die Buszuweisungssteuerung erfolgt über zwei Anschlußstifte, die den Eingang $\overline{BGACK}$ ersetzen.
- Die Steuerung für Unterbrechungen bedient sich zweier Eingänge anstatt drei. Die Leitungen $\overline{IPL0}$ und $\overline{IPL2}$ sind verbunden, nur $\overline{IPL1}$ ist separat.
- Der Befehlsvorrat ist fast der gleiche wie beim 68000. Lediglich sechs Befehle sind nicht vorhanden.

LESEZYKLUS

Der Lesezyklus für ein Byte ist identisch mit dem des 68000, hier aber mit dem Adreßbus A0–A19 anstatt A1–A23 und mit $\overline{DS}$ anstatt $\overline{UDS}$ und $\overline{LDS}$.

Das Lesen eines Wortes ist jedoch in zwei Zyklen aufgeteilt:

Lesezyklus des ersten Bytes
Lesezyklus des zweiten Bytes

Das Ablauf- und das Zeitdiagramm dieses Zyklus sehen Sie in Abb. 7.2 bzw. 7.3.

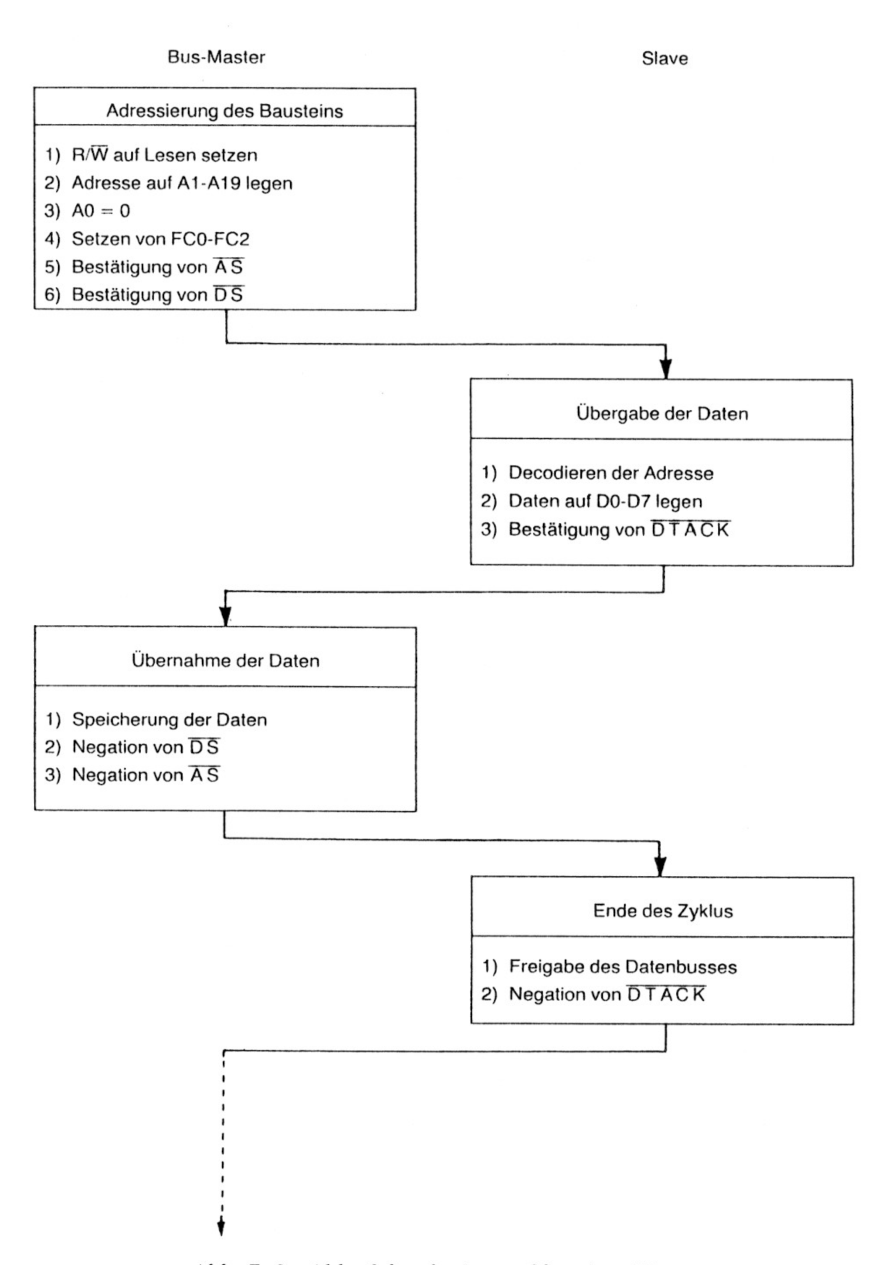


Abb. 7..2: Ablaufplan des Lesezyklus eines Wortes

DER BEFEHLSSATZ

Die Adressierungsarten sind die gleichen wie beim 68000. Lediglich der Befehlssatz unterscheidet sich geringfügig, da sechs Befehle fehlen. Und zwar handelt es sich um die Befehle:

ANDI mit SR
ANDI mit CCR
EORI mit SR
EORI mit CCR
ORI mit SR
ORI mit CCR

DER MC68010

ALLGEMEINES

Das Ziel des nun folgenden Kapitels ist es, die Verbesserungen dieses Prozessors gegenüber dem MC68000 aufzuzeigen.

Der MC68010, ein 16-Bit-Mikroprozessor, stellt eine Erweiterung des MC68000 dar. Er bietet die Möglichkeiten der virtuellen Maschine und des virtuellen Speichers, bleibt aber gleichzeitig kompatibel zum 68000. Eine sinnvolle Forderung für Mitglieder einer Familie ist nämlich, daß die für die anderen Mitglieder dieser Familie geschriebene Software ohne Änderung lauffähig ist.

Die wichtigsten Merkmale des 68010 sind die folgenden:

- direktes 16-Megabyte-Adressierungsspektrum
- 17 32-Bit-Daten- und Adreßregister
- Unterstützung von virtuellem Speicher und virtueller Maschine
- 57 Befehlstypen
- 14 Adressierungsarten

Konzept des virtuellen Speichers

Ursprünglich ist das Konzept des virtuellen Speichers aus der Notwendigkeit entstanden, kostspielige Speicherkapazität optimal zu nutzen. Ein System mit virtuellem Speicher ermöglicht nämlich dem Benutzer, Programme mit logischen (virtuellen) Adressen auszuführen, ohne sich um deren physikalische Existenz kümmern zu müssen.

Ein Speicherorganisationssystem, das die Hardware und die Software einschließt, paßt die Adressen des Programmierers dem geringen physischen Speicher an. Der Anwender benötigt keine Kenntnis über den Speicherplatz, an dem sein Programm abgelegt wurde.

Die physischen und virtuellen Adreßräume werden in Seiten aufgeteilt. Wird ein Zugriff zu einer Adresse auf einer Seite durchgeführt, die im physischen Speicher nicht vorhanden ist, so wird eine Ausnahmeverarbeitung eingeleitet. Das virtuelle Speichersystem kann diesen Fehler beheben, indem es die Seite ausfindig macht und sie durch eine der Seiten des physischen Speichers ersetzt.

Während dieses Ersetzungsvorgangs ist der Prozessor frei, um andere Anwender zu bedienen, und bietet so außerdem noch die Möglichkeit des Multiprogramms. Nach der Behebung des Fehlers kann der Prozessor sodann die Ausführung des unterbrochenen Programms wieder aufnehmen.

Ein Prozessor, der eine virtuelle Speicherorganisation ermöglicht, muß also in der Lage sein, die folgenden drei Basisfunktionen auszuführen: Fehlererkennung, Retten der gesamten Information, die notwendig ist, um den Fehler zu beseitigen, und Durchführen einer Ausnahmeprozedur und danach Wiederherstellung des alten Zustands und Wiederaufnahme der Verarbeitung.

Begriff der virtuellen Maschine

Bei diesem System können die Anwenderprogramme auf Elemente wie Plattenspeicher-Kontroller, Bildschirm-Kontroller, Drucker etc. zugreifen, obwohl es sie im System gar nicht gibt.

Eine typische Anwendung dafür ist die Fehlerbeseitigung in eine Software, die noch nicht funktionsfähig ist.

INTERNE STRUKTUR DES MC68010

Die interne Struktur des MC68010 ist in Abb. 7.13 dargestellt.

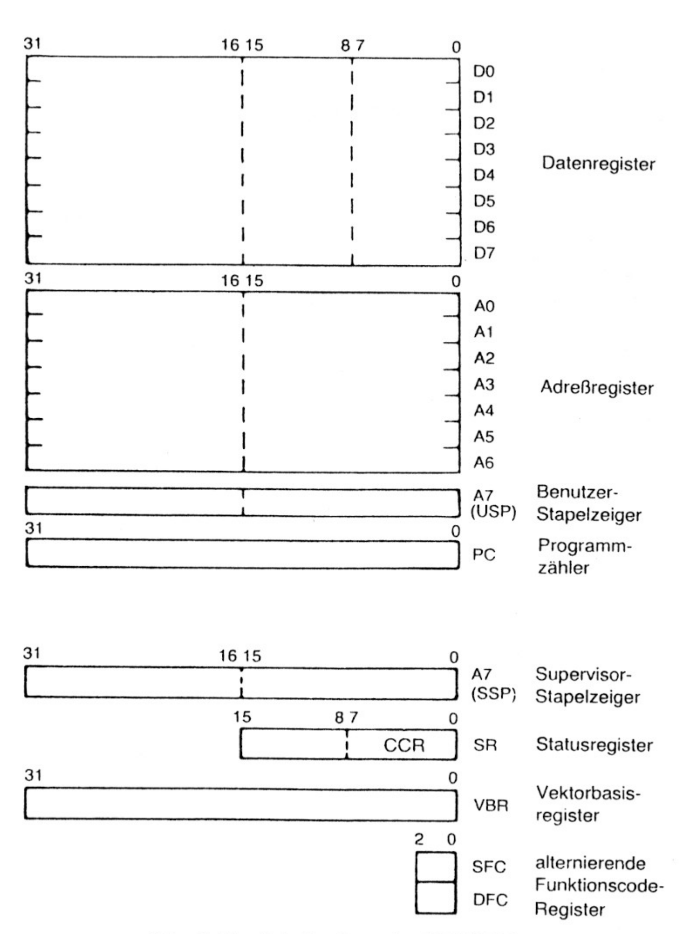


Abb. 7.13: Die Register des MC68010

Das Vektorbasisregister VBR

Dieses 32-Bit-Register kann dank des privilegierten Befehls MOVEC geladen und gelesen werden. Es wird bei der Errechnung der Vektoradresse eingesetzt.

Die alternierenden Funktionscode-Register SFC und DFC

Diese 3-Bit-Register können durch den MOVEC-Befehl geladen und gelesen werden. Der Befehl MOVES benutzt nur diese Register. Sie

erlauben dem Supervisor-System den Zugriff zu anderen adressierbaren Bereichen als den Supervisor-Daten.

DIE ANSCHLÜSSE DES MC68010

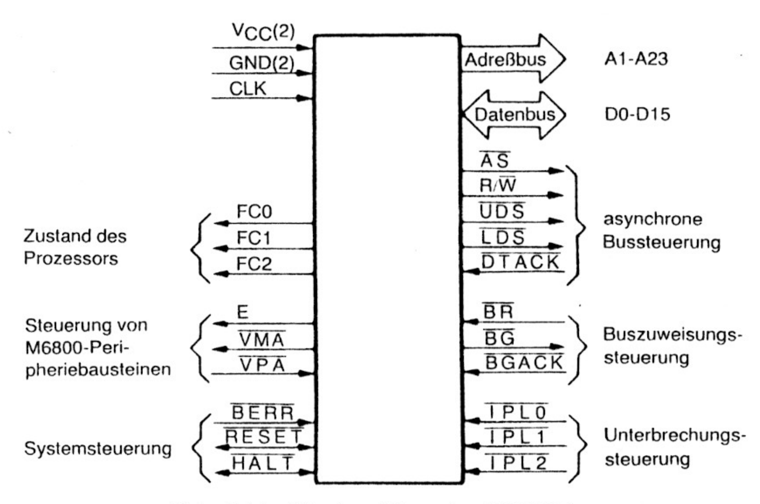


Abb. 7.14: Die Anschlüsse des MC68010

Der MC 68010 hat genau die gleichen Anschlüsse wie der MC68000. Es treten nur ein paar Varianten bei den Zuständen bestimmter Signale (FC0–FC2) und Prozeduren ($\overline{\text{BERR}}$) auf.

Eine Tabelle der Funktionscodes ist in Abb. 7.15 zu sehen.

FC2	FC1	FC0	Zyklustyp
L	L	L	reserviert
L	L	H	Anwender-Daten
L	H	L	Anwender-Programm
L	H	H	reserviert
H	L	L	reserviert
H	L	H	Supervisor-Daten
H	H	L	Supervisor-Programm
H	H	H	CPU-Bereich

Abb. 7.15: Tabelle der Funktionscodes

Die Ports werden auf die oberen Bits des Datenbusses geladen, z. B.:

- ein 16-Bit-Port D16–D31;
- ein 8-Bit-Port D24–D31.

Diese Transfers werden von den Signalen DSack0 (Data Transfer and Size acknowledge) und DSack1 gesteuert.

Diese zwei Eingangssignale funktionieren ähnlich wie DTACK; sie geben an, ob es sich um 8-, 16- oder 32-Bit-Übertragungen handelt.

Parallel dazu geben die beiden Signale Siz0 und Siz1 die noch zu übertragende Bytezahl an.

Somit zeigen während des Lesevorgangs eines Doppelwortes, beim ersten Zyklus diese Signale noch 4 Bytes an. Wenn wir es mit einem 8-Bit-Port zu tun haben, sind nach dem zweiten Zyklus noch 3 Bytes zu übertragen.

Ungerade Adressen auf dem Bus

Der MC68020 läßt Wort- und Doppelwortadressierung (auf der Operandenebene) mit ungeraden Adressen zu. Ein Adreßfehler wird nur im Fall eines an einer ungeraden Adresse plazierten Befehls erzeugt.

DIE PROGRAMMIERUNG DES MC68020

Die Programmierung wird durch die Flexibilität der Befehle vereinfacht. Wir haben z. B. folgende Verbesserungen:

- zusätzliche Befehle, die auf höhere Programmiersprachen ausgerichtet sind;
- Befehle, die beim MC68010 nur mit 8 oder 16 Bits arbeiten, sind hier mit 32 Bits möglich;
- elegante Adressierung bei Verzweigungen: 32-Bit-Adreßdistanzwerte sind erlaubt;
- besondere Adressierungsarten, die z. B. Verschiebung in ein Indexregister oder zusätzliche Parameter wie z. B. den Rangordnungsfaktor erlauben;
- ungefähr 16 zusätzliche Befehle wie TRAPcc, der eine Erweiterung von TRAPV ist.

Anhang **A**

Ausführungszeiten der Befehle des MC 68000

Berechnung der effektiven Adresse

	Adressierungart	Byte, Wort	Doppelwort
	Register		
Dn	Datenregister direkt	**0**(0/0)	**0**(0/0)
An	Adreßregister direkt	**0**(0/0)	**0**(0/0)
	Speicher		
(An)	Adreßregister indirekt	**4**(1/0)	**8**(2/0)
(An) +	Adreßregister indirekt mit Postinkrementierung	**4**(1/0)	**8**(2/0)
− (An)	Adreßregister indirekt mit Predekrementierung	**6**(1/0)	**10**(2/0)
d(An)	Adreßregister indirekt mit Adreßdistanzwert	**8**(2/0)	**12**(3/0)
d(An, $_{Ix}$)*	Adreßregister indirekt mit Index	**10**(2/0)	**14**(3/0)
$_{xxx}$W	absolut kurz	**8**(2/0)	**12**(3/0)
$_{xxx}$L	absolut lang	**12**(3/0)	**16**(4/0)
d(PC)	Programmzähler mit Adreßdistanzwert	**8**(2/0)	**12**(3/0)
d(PC, $_{Ix}$)*	Programmzähler mit Index	**10**(2/0)	**14**(3/0)
#$_{xxx}$	unmittelbar	**4**(1/0)	**8**(2/0)

* Die Länge des Indexregisters (ix) beeinflußt nicht die Ausführungszeit

MOVE (Byte und Wort)

Quelle	Ziel								
	Dn	An	(An)	(An) +	− (An)	d(An)	d(An, ix)*	xxx.W	xxx.L
Dn	**4**(1/0)	**4**(1/0)	**8**(1/1)	**8**(1/1)	**8**(1/1)	**12**(2/1)	**14**(2/1)	**12**(2/1)	**16**(3/1)
An	**4**(1/0)	**4**(1/0)	**8**(1/1)	**8**(1/1)	**8**(1/1)	**12**(2/1)	**14**(2/1)	**12**(2/1)	**16**(3/1)
(An)	**8**(2/0)	**8**(2/0)	**12**(2/1)	**12**(2/1)	**12**(2/1)	**16**(3/1)	**18**(3/1)	**16**(3/1)	**20**(4/1)
(An) +	**8**(2/0)	**8**(2/0)	**12**(2/1)	**12**(2/1)	**12**(2/1)	**16**(3/1)	**18**(3/1)	**16**(3/1)	**20**(4/1)
− (An)	**10**(2/0)	**10**(2/0)	**14**(2/1)	**14**(2/1)	**14**(2/1)	**18**(3/1)	**20**(3/1)	**18**(3/1)	**22**(4/1)
d(An)	**12**(3/0)	**12**(3/0)	**16**(3/1)	**16**(3/1)	**16**(3/1)	**20**(4/1)	**22**(4/1)	**20**(4/1)	**24**(5/1)
d(An, $_{Ix}$)*	**14**(3/0)	**14**(3/0)	**18**(3/1)	**18**(3/1)	**18**(3/1)	**22**(4/1)	**24**(4/1)	**22**(4/1)	**26**(5/1)
$_{xxx}$W	**12**(3/0)	**12**(3/0)	**16**(3/1)	**16**(3/1)	**16**(3/1)	**20**(4/1)	**22**(4/1)	**20**(4/1)	**24**(5/1)
$_{xxx}$L	**16**(4/0)	**16**(4/0)	**20**(4/1)	**20**(4/1)	**20**(4/1)	**24**(5/1)	**26**(5/1)	**24**(5/1)	**28**(6/1)
d(PC)	**12**(3/0)	**12**(3/0)	**16**(3/1)	**16**(3/1)	**16**(3/1)	**20**(4/1)	**22**(4/1)	**20**(4/1)	**24**(5/1)
d(PC, $_{Ix}$)*	**14**(3/0)	**14**(3/0)	**18**(3/1)	**18**(3/1)	**18**(3/1)	**22**(4/1)	**24**(4/1)	**22**(4/1)	**26**(5/1)
#$_{xxx}$	**8**(2/0)	**8**(2/0)	**12**(2/1)	**12**(2/1)	**12**(2/1)	**16**(3/1)	**18**(3/1)	**16**(3/1)	**20**(4/1)

* Die Länge des Indexregisters beeinflußt nicht die Ausführungszeit

MOVE (Doppelwort)

Quelle	Ziel								
	Dn	An	(An)	(An) +	− (An)	d(An)	d(An, ix)*	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	12(1/2)	12(1/2)	14(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
An	4(1/0)	4(1/0)	12(1/2)	12(1/2)	14(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
(An)	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
(An) +	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
− (An)	14(3/0)	14(3/0)	22(3/2)	22(3/2)	22(3/2)	26(4/2)	28(4/2)	26(4/2)	30(5/2)
d(An)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
d(An, Ix)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
xxx W	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
xxx L	20(5/0)	20(5/0)	28(5/2)	28(5/2)	28(5/2)	32(6/2)	34(6/2)	32(6/2)	36(7/2)
d(PC)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(5/2)
d(PC, Ix)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
#xxx	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)

* Die Größe des Indexregisters beeinflußt nicht die Ausführungszeit

Gebräuchliche Befehle

Befehl	Größe	op <ea>, Ant	op <ea>, Dn	op Dn, <M>
ADD	Byte, Wort	8(1/0) +	4(1/0) +	8(1/1) +
	Doppelwort	6(1/0) +**	6(1/0) +**	12(1/2) +
AND	Byte, Wort	–	4(1/0) +	8(1/1) +
	Doppelwort	–	6(1/0) +**	12(1/2) +
CMP	Byte, Wort	6(1/0) +	4(1/0) +	–
	Doppelwort	6(1/0) +	6(1/0) +	–
DIVS	–	–	158(1/0) +*	–
DIVU	–	–	140(1/0) +*	–
EOR	Byte, Wort	–	4(1/0)***	8(1/1) +
	Doppelwort	–	8(1/0)***	12(1/2) +
MULS	–	–	70(1/0) +*	–
MULU	–	–	70(1/0) +*	–
OR	Byte, Wort	–	4(1/0) +	8(1/1) +
	Doppelwort	–	6(1/0) +**	12(1/2) +
SUB	Byte, Wort	8(1/0) +	4(1/0) +	8(1/1) +
	Doppelwort	6(1/0) +**	6(1/0) +**	12(1/2) +

Beachten Sie:

+ Addieren Sie die Berechnungszeit für die effektive Adresse
t Nur Wort und Doppelwort
* Zeigt Maximalwert an
** Die Basiszeit von 6 Taktzyklen erhöht sich auf 8, wenn die Adressierungsart Register direkt oder unmittelbar ist (Zeit zur Berechnung der effektiven Adresse muß zusätzlich addiert werden.
*** Die einzige verfügbare effektive Adressierungsart ist Datenregister direkt.

DIVS, DIVU – Der Dividier-Algorithmus, den der MC68000 verwendet, bewirkt, daß sich die Zeiten für den günstigsten und den ungünstigsten Fall nur um 10% unterscheiden.

MULS, MULU – Der Multiplizier-Algorithmus benötigt 38+2n Taktzyklen, wobei n definiert ist als

MULU n= Anzahl der Einsen in der <EA>

MULS n = Zusammenfügen der <EA> und einer 0 als LSB. n ist die Anzahl der 10- oder 01-Muster in der 17-Bit-Quelle. Der ungünstigste Fall tritt dann auf, wenn die Quelle $5555 ist.

Unmittelbare Befehle

Befehl	Größe	op #, Dn	op #, An	op #, M
ADDI	Byte, Wort	**8**(2/0)	–	**12**(2/1) +
	Doppelwort	**16**(3/0)	–	**20**(3/2) +
ADDQ	Byte, Wort	**4**(1/0) +	**8**(1/0)*	**8**(1/1) +
	Doppelwort	**8**(1/0)	**8**(1/0)	**12**(1/2) +
ANDI	Byte, Wort	**8**(2/0)	–	**12**(2/1) +
	Doppelwort	**16**(3/0)	–	**20**(3/1) +
CMPI	Byte, Wort	**8**(2/0)	–	**8**(2/0) +
	Doppelwort	**14**(3/0)	–	**12**(3/0) +
EORI	Byte, Wort	**8**(2/0)	–	**12**(2/1) +
	Doppelwort	**16**(3/0)	–	**20**(3/2) +
MOVEQ	Doppelwort	**4**(1/0)	–	–
ORI	Byte, Wort	**8**(21/0)	–	**12**(2/1) +
	Doppelwort	**16**(3/0)	–	**20**(3/2) +
SUBI	Byte, Wort	**8**(2/0)	–	**12**(2/1) +
	Doppelwort	**16**(3/0)	–	**20**(3/2) +
SUBQ	Byte, Wort	**4**(1/0)	**8**(1/0)*	**8**(1/1) +
	Doppelwort	**8**(1/0)	**8**(1/0)	**12**(1/2) +

\+ Addieren Sie die Berechnungszeit für die effektive Adresse

* Nur Wort

Befehle, die nur einen einzigen Operanden verwenden

Befehl	Größe	Register	Speicher
CLR	Byte, Wort	**4**(1/0)	**8**(1/1) +
	Doppelwort	**6**(1/0)	**12**(1/2) +
NBCD	Byte	**6**(1/0	**8**(1/1) +
NEG	Byte, Wort	**4**(1/0)	**8**(1/1) +_
	Doppelwort	**6**(1/0)	**12**(1/2) +
NEGX	Byte, Wort	**4**(1/0)	**8**(1/1) +
	Doppelwort	**6**(1/0)	**12**(1/2) +
NOT	Byte, Wort	**4**(1/0)	**8**(1/1) +
	Doppelwort	**6**(1/0)	**12**(1/2) +
SCC	Byte, unwahr	**4**(1/0)	**8**(1/1) +
	Byte, wahr	**6**(1/0)	**8**(1/1) +
TAS	Byte	**4**(1/0)	**10**(1/1) +
TST	Byte, Wort	**4**(1/0)	**4**(1/0) +
	Doppelwort	**4**(1/0)	**4**(1/0)

\+ Addieren Sie die Berechnungszeit für die effektive Adresse

Schiebe- und Rotierbefehle

Befehl	Größe	Register	Speicher
ASR, ASL	Byte, Wort	**6 + 2n**(1/0)	**8**(1/1) +
	Doppelwort	**8 + 2n**(1/0)	–
LSR, LSL	Byte, Wort	**6 + 2n**(1/0)	**8**(1/1) +
	Doppelwort	**8 + 2n**(1/0)	–
ROR, ROL	Byte, Wort	**6 + 2n**(1/0)	**8**(1/1) +
	Doppelwort	**8 + 2n**(1/0)	–
ROXR, ROXL	Byte, Wort	**6 + 2n**(1/0)	**8**(1/1) +
	Doppelwort	**8 + 2n**(1/0)	–

\+ Addieren Sie die zur Berechnung der effektiven Adresse benötigte Zeit
n ist die Anzahl der Bitpositionen, um die verschoben wird

Bitmanipulation

Befehl	Größe	Dynamisch		Statisch	
		Register	Speicher	Register	Speicher
BCHG	Byte	–	**8**(1/1) +	–	**12**(2/1) +
	Doppelwort	**8**(1/0)*	–	**12**(2/0)*	–
BCLR	Byte	–	**8**(1/1) +	–	**12**(2/1) +
	Doppelwort	**10**(1/0)*	–	**14**(2/0)*	–
BSET	Byte	–	**8**(1/1) +	–	**12**(2/1) +
	Doppelwort	**8**(1/0)*	–	**12**(2/0)*	–
BTST	Byte	–	**4**(1/1) +	–	**8**(2/1) +
	Doppelwort	**6**(1/0)*	–	**10**(2/0)*	–

\+ Addieren Sie die zur Berechnung der effektiven Adresse benötigte Zeit
* Zeigt Maximalwert an

Bedingte Befehle

Befehl	Adreßdistanzwert	Verzweigung ausgeführt	Verzweigung nicht ausgeführt
B_{CC}	Byte	**10**(2/0)	**8**(1/0)
	Wort	**10**(2/0)	**12**(2/0)
BRA	Byte	**10**(2/0)	–
	Wort	**10**(2/0)	–
BSR	Byte	**18**(2/2)	–
	Wort	**18**(2/2)	–
DB_{CC}	CC wahr	–	**12**(2/0)
	CC falsch	**10**(2/0)	**14**(2/0)

\+ Addieren Sie die zur Berechnung der effektiven Adresse benötigte Zeit
* Zeigt Maximalwert an

JMP, JSR, LEA, PEA und MOVEM

Befehl	Größe	(An)	(An)+	−(An)	d(An)	d(An, ix)+	xxx.W	xxx.L	d(PC)	d(PC, ix)*
JMP	–	**8**(2/0)	–	–	**10**(2/0)	**14**(3/0)	**10**(2/0)	**12**(3/0)	**10**(2/0)	**14**(3/0)
JSR	–	**16**(2/2)	–	–	**18**(2/2)	**22**(2/2)	**18**(2/2)	**20**(3/2)	**18**(2/2)	**22**(2/2)
LEA	–	**4**(1/0)	–	–	**8**(2/0)	**12**(2/0)	**8**(2/0)	**12**(3/0)	**8**(2/0)	**12**(2/0)
PEA	–	**12**(1/2)	–	–	**16**(2/2)	**20**(2/2)	**16**(2/2)	**20**(3/2)	**16**(2/2)	**20**(2/2)
MOVEM M→R	Wort	**12+4n** (3+n/0)	**12+4n** (3+n/0)	– –	**16+4n** (4+n/0)	**18+4n** (4+n/0)	**16+4n** (4+n/0)	**20+4n** (5+n/0)	**16+4n** (4+n/0)	**18+4n** (4+n/0)
	Doppelwort	**12+8n** (3+2n/0)	**12+8n** (3+2n/0)	– –	**16+8n** (4+2n/0)	**18+8n** (4+2n/0)	**16+8n** (4+2n/0)	**20+8n** (5+2n/0)	**16+8n** (4+2n/0)	**18+8n** (4+2n/0)
MOVEM R→M	Wort	**8+4n** (2/n)	– –	**8+4n** (2/n)	**12+4n** (3/n)	**14+4n** (3/n)	**12+4n** (3/n)	**16+4n** (4/n)	– –	– –
	Doppelwort	**8+8n** (2/2n)	– –	**8+8n** (2/2n)	**12+8n** (3/2n)	**14+8n** (3/2n)	**12+8n** (3/2n)	**16+8** (4/2n)	– –	– –

n ist die Anzahl zu bewegender Registerinhalte

* Die Größe des Indexregisters (ix) beeinflußt nicht die Ausführungszeit

Mehrfachgenaue Befehle

Befehl	Größe	Op Dn, Dn	Op M, M
ADDX	Byte, Wort	**4**(1/0)	**18**(3/1)
	Doppelwort	**8**(1/0)	**30**5/2)
CMPM	Byte, Wort	–	**12**(3/0)
	Doppelwort	–	**20**(5/0)
SUBX	Byte, Wort	**4**(1/0)	**18**(3/1)
	Doppelwort	**8**(1/0)	**30**5/2)
ABCD	Byte	**6**(1/0)	**18**(3/1)
SBCD	Byte	**6**(1/0)	**18**(3/1)

Verschiedene Befehle

Befehl	Größe	Register	Speicher
ANDI to CCR	Byte	**20**(3/0)	–
ANDI to SR	Wort	**20**(3/0)	–
CHK	–	**10**(1/0)+	–
EORI to CCR	Byte	**20**(3/0)	–
EORI to SR	Wort	**20**(3/0)	–
ORI to CCR	Byte	**20**(3/0)	–
ORI to SR	Wort	**20**(3/0)	–
MOVE from SR	–	**6**(1/0)	**8**(1/1)+
MOVE to CCR	–	**12**(2/0)	**12**(2/0)+
MOVE to SR	–	**12**(2/0)	**12**(2/0)+
EXG	–	**6**(1/0)	–
EXT	Wort	**4**(1/0)	–
	Doppelwort	**4**(1/0)	–
LINK	–	**16**(2/2)	–
MOVE from USP	–	**4**(1/0)	–
MOVE to USP	–	**4**(1/0)	–
NOP	–	**4**(1/0)	–
RESET	–	**132**(1/0)	–
RTE	–	**20**(5/0)	–
RTR	–	**20**(5/0)	–
RTS	–	**16**(4/0)	–
STOP	–	**4**(0/0)	–
SWAP	–	**4**(1/0)	–
TRAPV	–	**4**(1/0)	–
UNLK	–	**12**(3/0)	–

+ Addieren Sie die zur Berechnung der effektiven Adresse benötigte Zeit

MOVEP

Befehl	Größe	Register → Speicher	Speicher → Register
MOVEP	Wort	**16**(2/2)	**16**(4/0)
	Doppelwort	**24**(2/4)	**24**(6/0)

Ausnahme-Prozeduren

Ausnahme	Zyklen
Adreßfehler	**50**(4/7)
Busfehler	**50**(4/7)
CHK-Befehl	**44**(5/4)+
Division durch Null	**42**(5/4)
Illegaler Befehl	**34**(4/3)
Unterbrechung	**44**(5/3)*
Privilegverletzung	**34**(4/3)
$\overline{\text{RESET}}$**	**40**(6/0)
Trace	**34**(4/3)
TRAP-Befehl	**38**(4/4)
TRAPV-Befehl	**34**(4/3)

+ Addieren Sie die zur Berechnung der effektiven Adresse benötigte Zeit

* Es wird angenommen, daß der Unterbrechungserkennungszyklus vier Taktzyklen dauert.

** Zeit vor der Erkennung, daß $\overline{\text{RESET}}$ und $\overline{\text{HALT}}$ aktiv sind, bis zum Beginn der Befehlsausführung

Ausnahme-Prozeduren

Ausnahme	
Adreßfehler	**126**(4/26)
Breakpoint-Befehl	**42**(5/4)
Busfehler	**126**(4/26)
CHK-Befehl**	**44**(5/4)+
Division durch Null	**44**(5/4)
Illegaler Befehl	**38**(4/4)
Unterbrechung	**46**(5/4)
illegaler Cr**	**46**(5/4)
Privilegverletzung	**38**(4/4)
Reset***	**40**(6/0)
RTE, illegales Format	**50**(7/4)
RTE, illegale Revision	**70**(12/4)
Trace	**38**(4/4)
TRAP-Befehl	**38**(4/4)
TRAPV-Befehl	**40**(5/4)

\+ Addieren Sie die zur Berechnung der effektiven Adresse benötigte Zeit

* Es wird angenommen, daß die Unterbrechungserkennungs- und Breakpoint-Zyklen vier Taktzyklen dauern

** Zeigt Maximalwert an.

*** Gibt die Zeit an, die von der ersten Erkennung von $\overline{RESET}$ und $\overline{HALT}$ bis zum Beginn der Ausführung des Befehls vergeht

Stichwortverzeichnis